НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки
(повне найменування інституту, факультету)

Кафедра обчислювальної техніки
(повна назва кафедри)

До захисту допущено:

Завідувач кафедри

_____ Сергій СТІРЕНКО
(підпис) (ім'я, прізвище)

«___»_____ 2020 р.

Дипломний проект

на здобуття ступеня бакалавра

за освітньо-професійною програмою *"Комп'ютерні системи та мережі"*

спеціальності *123 "Комп'ютерна інженерія"*

на тему *"Конструювання трафіку в програмно-конфігурованих мережах на основі технології Big Data"*

Виконав: студент *IV* курсу, групи *ІО-63*
Бабко Дмитро Сергійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник *асистент Калюжний Олександр Олегович* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант н. контроль *доц, д. т. н. Сімоненко В. П.* _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент *доц. каф. СПіСКС к.т.н., доц. Марія ОРЛОВА* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ - 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки
(повне найменування інституту, факультету)

Кафедра обчислювальної техніки
(повна назва кафедри)

Рівень вищої – освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма – «Комп'ютерні системи та мережі»

До захисту допущено:

Завідувач кафедри

_____ Сергій СТІРЕНКО
(підпис) (ім'я, прізвище)

«___»_____ 2020 р.

ЗАВДАННЯ

на дипломний проект студенту

Бабку Дмитру Сергійовичу

1. Тема проекту *"Конструювання трафіку в програмно-конфігурованих мережах на основі технології Big Data"*, керівник проекту *Калюжний Олександр Олегович асистент* , затверджені наказом по університету від *« 07 » травня* 2020р. № 1081-с

2. Термін подання студентом проекту *25.05.2020р.*

3. Вихідні дані до проекту: *технічне завдання, теоретичні дані, наукові публікації.*

4. Зміст пояснювальної записки: *огляд способів конструювання трафіку в програмно-конфігурованих мережах, спосіб конструювання трафіку в SDN на основі технології Big Data, проектування та розробка програмного продукту, моделювання та аналіз отриманих результатів.*

5. Консультант роботи, з вказівкою розділів роботи, які до них вносяться

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В.П.		

6. Дата видачі завдання 01.09.2019 року

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проекту (роботи)	Строк виконання етапів проекту(роботи)	Примітки
1	*Затвердження теми роботи*	*01.09.2019*	
2	*Вивчення та аналіз завдання*	*14.09.2019*	
3	*Розробка архітектури та загальної структури систем*	*12.10.2019*	
4	*Розробка структур окремих підсистем*	*26.10.2019*	
5	*Програмна реалізація системи*	*30.11.2020*	
6	*Оформлення пояснювальної записки*	*28.03.2020*	
7	*Передзахист*	*26.05.2020*	
8	*Захист*	*25.06.2020*	

Студентка Дмитро БАБКО _____
(підпис)

Керівник Олександр КАЛЮЖНИЙ _____
(підпис)

Анотація

Бакалаврська дипломна робота присвячена вирішенню проблеми конструювання трафіку в програмно-конфігурованих мережах в реальному часі та з використанням історичної інформації про мережу. Розглянуті способи моніторингу та аналізу трафіку вирішують проблему збору і використання статистичних даних. Розроблений програмний продукт забезпечує можливість моделювання системи конструювання трафіку на основі парадигм Big Data з метою подальшого впровадження способу в існуючі системи моніторингу та аналізу.

Annotation

The Bachelor's thesis is devoted to solving the problem of traffic engineering in software-defined networks in real time and using historical information about the network. The considered methods of traffic monitoring and analysis solve the problem of collecting and using statistical data. The developed software product provides the ability to model the traffic design system based on Big Data paradigms in order to further implement the method in existing monitoring and analysis systems.

№ рядка	Формат	Позначення	Найменування	Кільк. аркушів	№ прим.	Примітка
1	А4		Завдання на дипломний проект	2		
2	А4	ІАЛЦ.466454.001 ВД	Відомість дипломного проекту	1		
3	А4	ІАЛЦ.466454.002 ТЗ	Технічне завдання	3		
4	А4	ІАЛЦ.466454.003 ПЗ	Пояснювальна записка	60		
5	А4	ІАЛЦ.466454.004 Д1	Структурна схема пристрою	1		
6	А4	ІАЛЦ.466454.005 Д2	Функціональна схема	1		
7	А4	ІАЛЦ.466454.006 Д3	Принципова схема	1		
			апаратного забезпечення			
8	А4	ІАЛЦ.466454.007 Д4	Програмний код додатку	9		

Зм.	Арк.	№ документа	Підпис	Дата	ІАЛЦ.466454.001.ВД		
Розробив		Бабко Д.С.			Конструювання трафіку в програмно-конфігурованих мережах з використанням технології Big Data	Лит.	Лист. 1 / Аркушів 1
Перевірив		Калюжний О.О.					
Реценз.						НТУУ "КПІ", ФІОТ, ІО-63	
Н. Контр.		Сімоненко В. П.			Відомість дипломного проекту		
Затвердив							

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломної роботи
освітньо-кваліфікаційного рівня бакалавр

на тему: «Конструювання трафіку в програмно-конфігурованих мережах на основі технології Big Data»

Київ – 2020 року

ЗМІСТ

1. Найменування та область застосування

Розробка програми для реалізації моделювання системи конструювання трафіку на основі технології Big Data. Область застосування — корпоративні мережі.

2. Підстава для розробки

Підставою для розробки є індивідуальне завдання на реалізацію методу конструювання трафіку в програмно-конфігурованих мережах на основі технології Big Data, затверджене кафедрою Обчислювальної техніки ФІОТ НТУУ “КПІ ім. Ігоря Сікорського”.

3. Призначення розробки

Даний проект призначений для моделювання системи конструювання трафіка в програмно-конфігурованих мережах на основі технології Big Data, а також для демонстрації набутих навичок та закріплення отриманих знань.

4. Джерела розробки

Основними джерелами розробки є науково-технічна література по програмному забезпеченню, наукові публікації щодо конструювання трафіку в SDN, дослідження в області Big Data та результати застосування системи конструювання трафіку в корпоративних мережах.

5. Технічні вимоги

5.1 Вимоги до розроблюваного продукту

- Розробка інтерфейсу для моделювання мережі згідно заданим параметрам та генерація трафіку.
- Збереження історичної інформації в базі даних, реалізація доступу до даних системі моніторингу і аналізу трафіку

- Виконання симуляції роботи модельованої системи програмним шляхом для реалізації аналізу трафіку, демонстрація результатів роботи програми

5.2 Вимоги програмного забезпечення

- Ubuntu Linux 14.x+, Mac OS 10.8.3+, 10.9+, MS Windows 10
- Python 3+, відповідні бібліотеки networkx, sqlite3, matplotlib.
- Інтеграційне середовище розробки IntelliJ PyCharm, Jupiter Notebook

5.2 Вимоги апаратного забезпечення

- Для Windows: RAM 512 МБ; мінімальна вимога до процесора Pentium 2 266 МГц
- Для Mac OS X: Mac на базі процесора Intel, браузер Safari, Chrome
- Для Linux: браузер Chrome, Firefox

5.3 Вимоги до надійності

- Написання тестів для програмного забезпечення
- Проходження тестування у процесі роботи системи
- Строк використання — необмежений

6. Етапи розробки

1.	Дослідженні існуючих рішень	01.09.2019
2.	Постановка та аналіз технічного завдання	14.09.2019
3.	Розробка архітектури та основних модулів системи	12.10.2019
4.	Тестування окремих модулів та усієї системи	26.10.2019
5.	Налагодження програмного продукту	30.11.2020
6.	Написання документації до дипломної роботи	28.03.2020

Пояснювальна записка
до дипломного проекту
на тему: «Конструювання трафіку в програмно-конфігурованих мережах на основі технології Big Data»

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API	(Address Resolution Protocol) Прикладний програмний інтерфейс
BR	(Bytes Received) Байт отримано
BRT	(Bytes Received Throughput) Пропускна здатність по отриманих байтах
BT	(Bytes Transmitted) Байт передано
BTr	(Bytes Throughput) Пропускна здатність по байтах
BTT	(Bytes Transmitted Throughput) Пропускна здатність по переданих байтах
ER	(Entity-Relationship) Модель «сутність-відношення»
IP	(Internet Protocol) Інтернет протокол
OD	(Origin-Destination) старт – призначення
PR	(Bytes Received) Пакетів отримано
PRT	(Bytes Received Throughput) Пропускна здатність по отриманих пакетах
PT	(Bytes Transmitted) Пакетів передано
PTr	(Bytes Throughput) Пропускна здатність по пакетах
PTT	(Bytes Transmitted Throughput) Пропускна здатність по переданих пакет ах
SDN	(Software-defined Networks) Програмно-конфігуровані мережі
TM	(Traffic Matrix) Матриця трафіку

ВСТУП

За останні роки відбувся значний розвиток в експлуатації традиційних мереж. Зростання обсягів хмарних обчислень, Інтернету речей, надання мультимедійного контенту, що охоплює текст, звук і відео, значно впливають на складність конфігурації пристроїв, управління і обслуговування, впровадження інновацій та розробки нових послуг і сервісів. Для вирішення таких задач досліджуються технічні рішення, що підвищують гнучкість мережі, віддалюючись від закритої архітектури пристроїв і покращуючи програмованість.

Особливо спостерігається зростання популярності технології програмно-конфігурованих мереж (SDN). Такий підхід розділяє рівень керування від пристроїв передачі трафіка за допомогою централізованого керування замість звичайного розподіленого керування, а також дозволяє мережі бути програмно-конфігурованою, використовуючи відкриті і програмовані інтерфейси.

Окрім цього, швидкий розвиток та зростання пристроїв, підключених до певної мережі обумовлює зростання обсягу даних, що передаються мережею. Існуючі способи конструювання трафіку в SDN вразливі для опрацювання даних в режимі реального часу. Враховуючи наведені особливості архітектури та вимоги до сучасних мереж, доцільним є використання технологій Big Data для вирішення поставленої задачі.

Метою роботи є дослідження та модернізація використання технологій Big Data для конструювання трафіку в програмно-конфігурованих мережах.

Для досягнення мети поставлено наступні завдання:

- аналіз архітектури SDN і огляд існуючих підходів до конструювання трафіку;
- дослідження та модернізація способу конструювання трафіка з використанням сучасних парадигм Big Data;

- розробка програмного додатку системи моніторингу та аналізу трафіку, ілюстрація роботи додатку та аналіз отриманих результатів.

Об’єктом дослідження є процес конструювання трафіку в SDN. Предметом дослідження є процес формування матриці трафіку та генерації статистичних даних, використовуючи технології Big Data.

Для виконання поставлених задач використано методи теорії графів, методи імітаційного моделювання, методи візуалізації.

Наукова новизна отриманих результатів полягає в отриманні імітаційної моделі системи моніторингу та аналізу трафіку, що дозволяє працювати з великими обсягами даних майже в реальному часі; запропоновано використання лічильників фізичних помилок для модернізації розглянутого методу конструювання трафіку.

РОЗДІЛ 1. ОГЛЯД СПОСОБІВ КОНСТРУЮВАННЯ ТРАФІКУ В ПРОГРАМНО-КОНФІГУРОВАНИХ МЕРЕЖАХ

1.1. Загальний технологічний огляд SDN та Big Data

1.1.1. Структура мереж SDN

Сучасні апаратно-орієнтовані мережі створюють ряд проблем в області експлуатації і керування [1]. Основою інфраструктури такої мережі є пристрої пересилки, такі як комутатори та маршрутизатори. Вони мають два логічних компоненти для керування трафіком: рівень керування (що вирішує, як опрацьовувати мережевий трафік) і рівень даних (що направляє трафік на основі рішення площини керування) [2]. Така системи робить керування потоками трафіку досить ресурсномістким, особливо для масштабних мереж, так як будь-яка нова конфігурація має бути реалізована вручну на кожному з пристроїв. Окрім цього, встановлення конфігурацій також залежить від виробника: різні виробники мають власні набори правил для налаштування своїх пристроїв, що ще більш ускладнює процес керування.

У зв'язку з цим була запропонована нова мережева архітектура для подолання проблем і обмежень традиційних мереж. Програмно-конфігурована мережа (SDN*)* – мережа передачі даних, у якій рівень керування відділений від пристроїв передачі даних і реалізується програмно. SDN є динамічною, керованою, економічною і адаптованою архітектурою, що робить її ідеальною для високошвидкісної, динамічної природи сучасних додатків. Ця архітектура дозволяє напряму керувати програмуванням мережі і абстрагувати базову інфраструктуру для додатків і мережевих служб [3]. Структура SDN представлена на рис. 1.1 [4].

Рівень інфраструктури, або рівень даних, являє собою взаємопов'язані пристрої пересилки. Ці пристрої мають певний набір функцій, що визначають

дії, які мають бути виконані для вхідних пакетів даних (наприклад, пересилка на певні порти, пересилка в контролер, тимчасове чи постійне видалення).

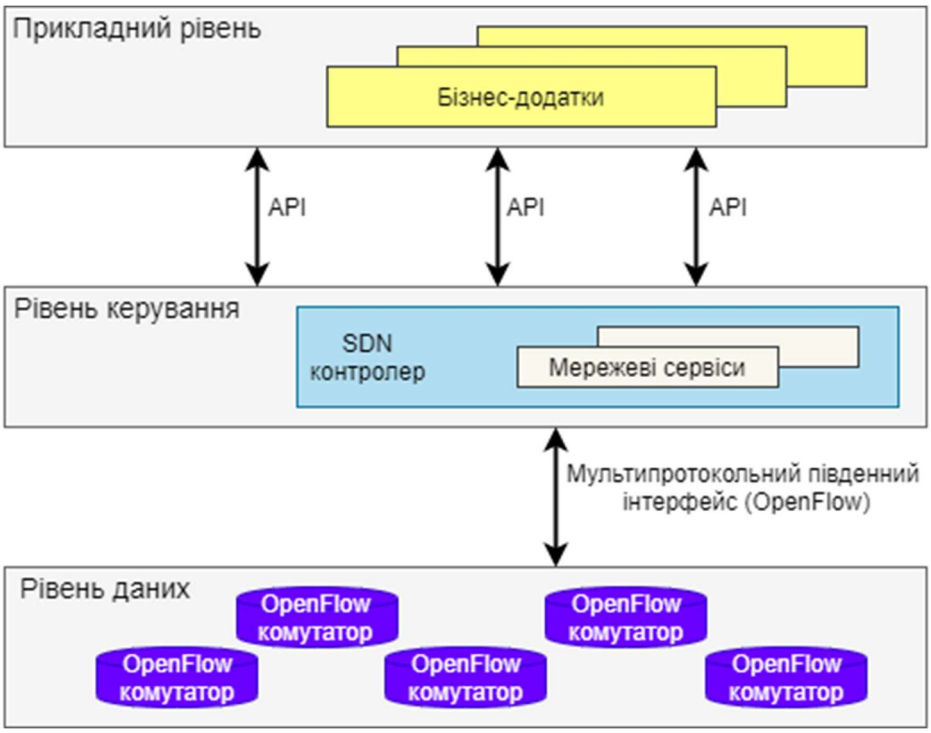


Рис. 1.1. Структура SDN

Мультипротокольний південний інтерфейс визначає множину інструкцій, що використовуються для взаємодії між елементами рівня керування і рівня даних. Він також визначає протокол зв'язку між рівнями.

Рівень керування представлений централізованим SDN-контролером. Він керує комутаторами для досягнення певних цілей додатків, а також забезпечує абстрактне представлення усієї мережевої структури.

Додатки виграють від розподілення між рівнями керування та даних. Вони можуть використовувати північний програмний інтерфейс для реалізації спеціальних операцій, таких як маршрутизація, моніторинг та конструювання трафіку.

Порівняно з традиційними мережами, SDN має такі переваги [5]:

1. Контролер має глобальне уявлення про топологію мережі, її стан і вимоги до додатків.

2. Рівень даних може бути запрограмовано динамічно для покращення розподілу мережевих ресурсів.
3. Зв’язок між контролером та комутаторами і роутерами не залежить від виробника пристрою.

Потік в загальних рисах є послідовністю пакетів, що проходять через мережу і спільно використовують множину значень полів заголовка, таких як однакові IP-адреси джерела та призначення, один і той самий ідентифікатор VLAN і одну й ту саму MAC-адресу [6]. Кожен новий потік потребує дозволу від контролера, що перевіряє потік на відповідність до мережевої політики. Контролер SDN визначає, які потоки можуть проходити через інфраструктурний рівень. Тобто, перший пакет кожного нового потоку надсилається рівнем даних до контролера для отримання дозволу на продовження в мережі, а також для отримання його маршруту по усім встановленим пристроям пересилки.

В програмно-конфігурованих мережах контролер має мережеву систему для забезпечення конструювання трафіку. Мережева операційна система надає програмний інтерфейс, через який здійснюється управління мережевими пристроями та постійне відстеження стану мережі [7].

SDN підтримує централізовану і розподілену архітектуру. У централізованій архітектурі один єдиний контролер повинен мати загальне представлення про топологію і відслідковувати всю інформацію про потоки і завантаженість кожного пристрою пересилки. Розподілена архітектура розділяє функцію керування між декількома контролерами (рис. 1.2) [8].

1.1.2. OpenFlow протокол

Для реалізації програмно-конфігурованної мережі мають бути виконані дві умови [6]:

1. Загальна логічна архітектура має бути в усіх мережевих пристроях, що керуються контролером. Навіть якщо різні виробники

реалізують логіку по-різному, контролер повинен бачити набір уніфікованих функцій.

2. Для зв’язку між контролером і пристроями має використовуватися стандартний протокол безпеки.

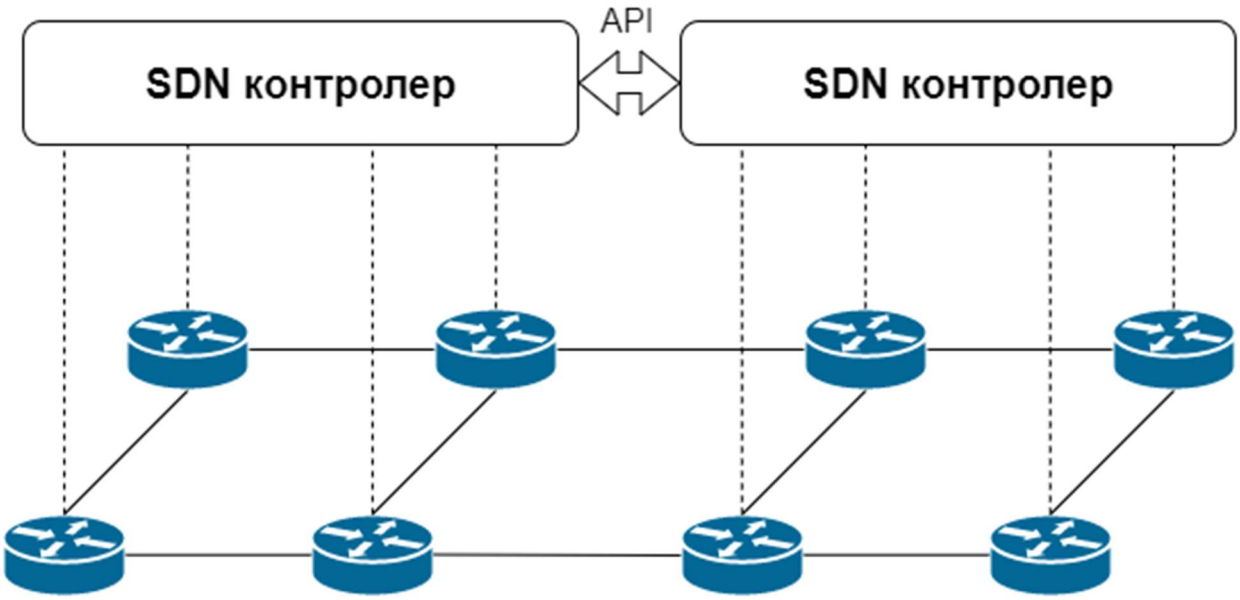


Рис. 1.2. Модель розподіленої архітектури SDN

OpenFlow відповідає цим критеріям, надаючи специфікації логічного формату функцій комутації мережі, а також є протоколом зв’язку між контролером та пристроями. Архітектура OpenFlow складається з комутаційного обладнання, керованого контролером OpenFlow і визначається Open Networking Foundation (ONF) у специфікації комутатора OpenFlow. Загальна модель комутатора OpenFlow представлена на рис. 1.3 [9].

У протоколі OpenFlow використовується концепція потоків для опису трафіку. Усі комутатори, що підтримують даний протокол, мають таблицю потоків, що складається з трьох частин: поля відповідності, лічильники і інструкції відносно пакетів. Поля відповідності допомагають у визначенні, до якого потоку відноситься кожний пакет, що проходить через комутатор. Поля лічильників дозволяють реалізувати моніторинг і отримувати статистику мережі в реальному часі [10]. У випадку, коли існує більше ніж одна таблиця, вони реалізовуються у вигляді конвеєру. Інструкції змінюють операцію

множини конвеєрної обробки. Лічильники оновлюються, коли пакети збігаються.

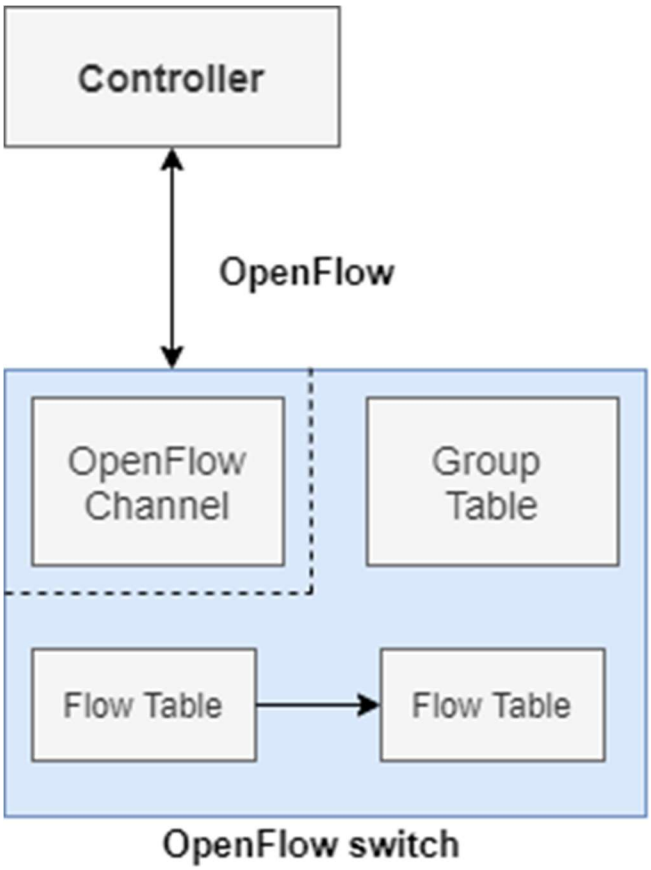


Рис. 1.3. Модель комутатора OpenFlow

Кожна таблиця потоків повинна містити запис о пропущених таблицях. Цей запис визначає дію у випадках, коли пакет не відповідає жодному запису в таблиці. У такому випадку пакет може бути перенаправлений на контролер, пропущений чи направлений у наступну таблицю. Запис пропущених таблиць в основному працює так само, як і будь-який інший запис потоку. Він не включений за замовчуванням, і контролер може його додати або видалити в будь-який час.

OpenFlow використовує потоки для ідентифікації мережевого трафіку. Для кожного потоку в кожному вузлі контролер може визначити відповідну інструкцію для опрацювання потоку. Інструкції можуть бути статично чи динамічно запрограмовані мережевою операційною системою (NOS) і

зберігаються в таблиці потоків. В загальному вигляді таблиця потоків виглядає, як зображено на рис. 1.4 [6].

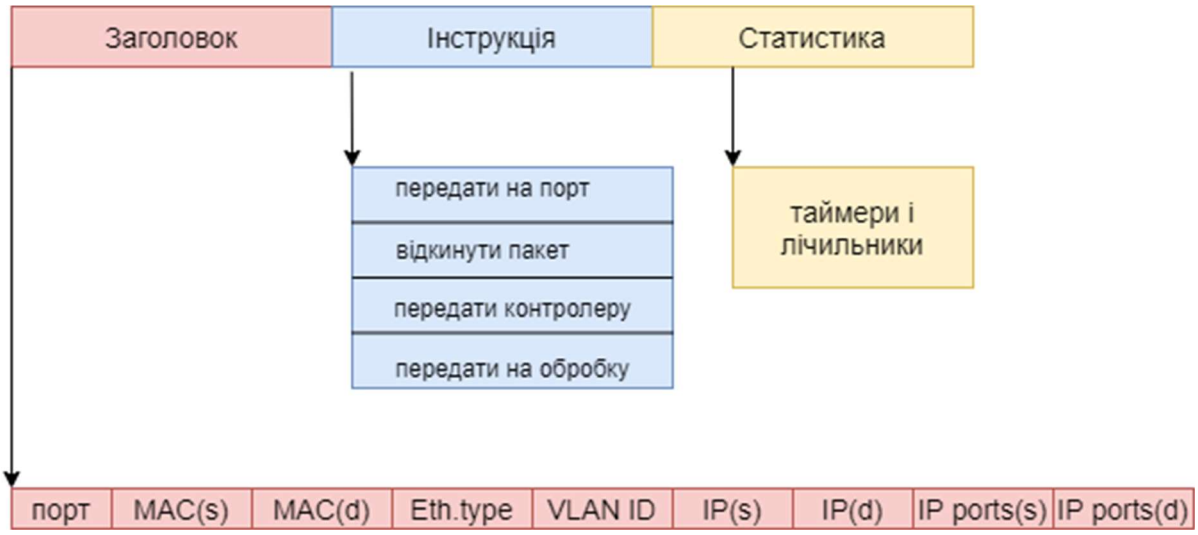


Рис. 1.4. Базовий вигляд таблиці потоків

Усі записи таблиці потоків містять заголовки для ідентифікації потоку. При надходженні нового пакету до комутатора визначається відповідний запис в таблиці потоків за визначеним алгоритмом. Крім цього, всі таблиці містять інструкції, необхідні до виконання: передача на порт, відкинути пакет, пересилка на контролер тощо.

Лічильники є важливою складовою таблиці потоків. Вони доступні для кожної таблиці потоків, запису потоку, порту, черги, групи тощо. Специфікація OpenFlow 1.4 визначає набір лічильників, перелік наведено в додатку А.

З точки зору окремого комутатор, потік – це послідовність пакетів, котра відповідає певному запису в таблиці потоків.

Протокол OpenFlow підтримує три типи повідомлень [6]:

Контролер-комутатор. Такі повідомлення потрібні для конфігурації та моніторингу стану комутаторів. Вони дозволяють контролеру керувати логічним станом комутатора, у тому числі його конфігурацією і контролювати деталі запису потоку і групової таблиці.

Асиметричні. Повідомлення надсилаються без запиту від контролера. Вони включають в себе різні повідомлення про стан контролера.

Симетричні. Повідомлення надсилаються без запиту від контролера або комутатора. До них належать: hello-повідомлення (при встановленні першого з'єднання між контролером і комутатором), echo-запити (для вимірювання затримки чи пропускної здатності каналу) тощо.

До однієї з переваг OpenFlow належить обернена сумісність з вже існуючими пристроями, адже даний протокол розроблявся з метою забезпечення максимальної підтримки існуючих апаратних рішень і мінімальних затрат на заміну обладнання на нове. Тому розрізняють також чисті OpenFlow комутатори та гібридні OpenFlow комутатори [11].

1.1.3. Огляд технологій Big Data

Термін Big Data належить до різних типів великих і неструктурованих наборів даних, які не можуть бути опрацьовані за допомогою звичайних систем обробки даних в необхідні строки [12]. Розмір – перша характеристика, що використовується для розуміння великих даних, але не єдина. Було запропоновано три виміри для визначення великих даних: об'єм, різноманітність і швидкість (англ. Three V's: Volume, Variety, Velocity) [13]. Об'єм відноситься до великої кількості даних, що створюються додатками. Різноманітність визначає наявність різних форматів даних у певному наборі. Різні технології забезпечують структуровані дані (електронні таблиці, реляційні бази даних), неструктуровані дані (текст, зображення, аудіо, відео) і напівструктуровані (XML). Поняття швидкості відноситься до швидкості генерації та аналізу даних.

В доповнення до вищеназваних характеристик існують також інші величини, пов'язані з великими даними: достовірність, мінливість і цінність. Під достовірністю мається на увазі наявність неточних і невизначених даних в певних джерелах. Мінливість відноситься до різних швидкостей передачі даних. Цінність виражає концепцію, згідно до якої дані, отримані в їх

первісному вигляді зазвичай мають низьку цінність по відношенню до їх об’єму.

Big Data – сукупність технологій, що створені для здійснення трьох базових операцій:

1. Опрацьовувати великі порівняно зі стандартними сценаріями обсяги даних.
2. Вміти працювати зі швидко зібраними даними в дуже великих обсягах, тобто працювати з динамічно зростаючим обсягом даних.
3. Вміти працювати з даними різної структури паралельно і в різних аспектах.

Виходячи з вищенаведених визначень, можна сформулювати основні принципи роботи з великими даними:

Горизонтальна масштабованість. Це є базовим принципом опрацювання великих даних. Зі зростанням обсягів даних необхідно збільшувати кількість обчислювальних вузлів, по котрим розподіляються ці дані, при цьому обробка має виконуватись без погіршення продуктивності.

Відмовостійкість. Цей принцип заснований на попередньому. Так як обчислювальний кластер може містити в собі велику кількість вузлів (іноді десятки тисяч) і їх кількість, що не виключено, може збільшуватись, зростає і ймовірність виходу машин з ладу. Методи роботи з великими даними мають враховувати можливість таких ситуацій і передбачати превентивні міри.

Локальність даних. Зазвичай дані розподілені по значній кількості вузлів, тому, якщо вони фізично знаходяться на одному сервері, а опрацьовуються на іншому, витрати на передачу даних можуть бути невиправдано великими. Бажано виконувати обробку даних на тій самій машині, на якій вони зберігаються.

Ці принципи відрізняються від звичних традиційних, централізованих, вертикальних моделей збереження добре структурованих даних. Відповідно,

для роботи з великими даними створюються новітні технологічні підходи [14].

Пакетна обробка і обробка в реальному часі – два різні типи обробки великих даних. Вибір більш відповідного типу обробки для конкретного додатку залежить від типу і джерел даних, необхідного часу обробки і необхідності виконання негайних дій [15]. Обробка в реальному часі містить безперервний ввід, обробку і вивід даних і необхідна у випадках, коли потрібно вжити термінових заходів на основі опрацьованих даних.

Слід зауважити, що існують помилки в організації керування даними в режимі реального часу, одна з найважливіших це «обчислення в реальному часі це швидке обчислення» [16]. Швидка обробка не значить, що обмеження в реальному часі виконуються. Окрім цього, значення «обробки в реальному часі» суттєво залежить від додатку. Термін «обробка в реальному часі» більше пов'язаний з інтерактивністю даних, ніж з мілісекундним відгуком. Обробка запитів в реальному часі в середовищі великих даних має повертати результати за секунди чи хвилини, на відміну від пакетних рішень, що зазвичай завершуються за години або ж дні.

Метод, розглянути в роботі, може надавати результати за час, що вимірюється у секундах, тому його можна позиціонувати як забезпечення системи моніторингу в реальному часі.

MapReduce – парадигма програмування для паралельної обробки великих обсягів даних. Ідея MapReduce полягає в розподіленні даних і обробці по декількох вузлах для горизонтального масштабування. Згідно з назвою, модель MapReduce містить дві фази:

- *Map.* На цьому етапі декілька мапперів зчитують блоки даних і опрацьовують їх для отримання пар вигляду ключ-значення. Для створення проміжних пар можна виконувати декілька задач поспіль.

- *Reduce.* На даному етапі редуктори зчитують згенеровані пари ключ-значення і агрегують їх або групують по ключу, отримуючи кінцевий результат.

У процесі MapReduce дані, що обробляються спочатку, діляться на групи, і головний вузол назначає ці групи для робочих вузлів. Кожен робочий вузол опрацьовує власні розбиття для генерації проміжних файлів з парами ключ-значення. Робочі редуктори, що керуються головним вузлом, опрацьовують проміжні пари ключ-значення згідно до функцій групування для отримання кінцевого результату (рис. 1.5) [17].

Парадигма MapReduce знатна вирішувати значну кількість реальних задач. Складна задача може включати в себе декілька операцій MapReduce і може потребувати для виконання багато часу та ресурсів. Для вирішення цієї проблеми доступні два варіанти реалізації MapReduce: Hadoop і Spark. Ці інструменти відрізняються один від одного не тільки своїми операторами потоків даних, а і підходом до проміжних файлів. Hadoop зберігає проміжні файли на диску, тоді як Spark розміщує їх в пам'яті. Через цю відмінність пропонується використовувати Hadoop для пакетної обробки і Spark для обробки в реальному часі.

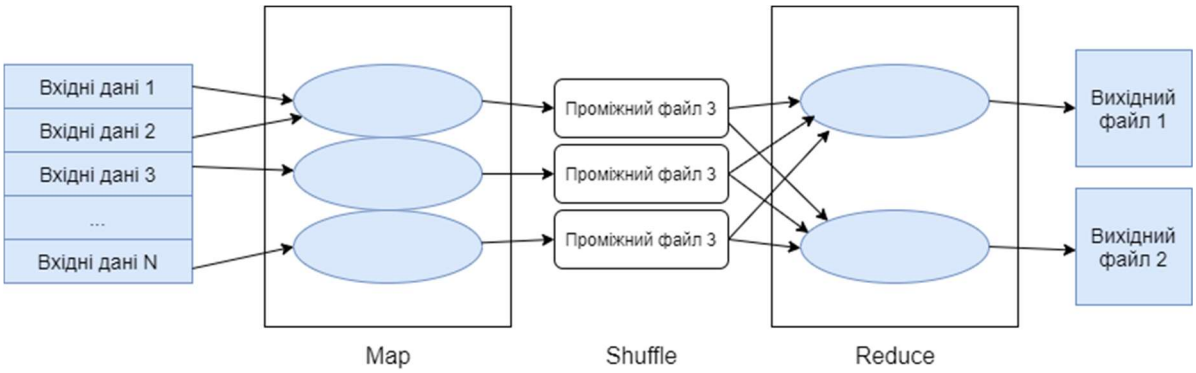


Рис.1.5. Процес MapReduce

Лямбда-архітектура (англ. Lambda Architecture) – система Big Data, що складається з ряду рівнів. Основна ідея лямбда-архітектури полягає в створенні системи, що може опрацьовувати і створювати різноманітні

представлення даних. Система, побудована на основі лямбда-архітектури, може опрацьовувати як потокові, так і пакетні дані. Для цього обробка даних фактично розділяється на два шляхи (рис. 1.6.) [8]:

«Холодний» шлях – пакетний рівень, де всі вхідні дані зберігаються в неопрацьованому вигляді і обробляються в пакетному режимі. Зазвичай пакетний рівень представлений озерами даних (data lakes), де знаходиться «сира» інформація, котра не опрацьовується, а лише доповнюється. Важливо, що саме цей рівень використовується для опрацювання даних по розкладу, тобто формуванні пакетних завдань.

«Гарячий» шлях – швидкісний рівень, або рівень прискорення, де дані аналізуються в режимі реального часу. Цей рівень забезпечує мінімальну затримку обробки з деякою втратою точності. Він являє собою сукупність складів даних, де в окремі представлення реального часу додається інформація з коротким життєвим циклом, щоб виключити дублювання даних.

Також в цій архітектурі виділяють *сервісний рівень,* котрий індексує пакетне представлення – результати роботи пакетного рівня для ефективного виконання запитів. За рахунок індексації і обробки вхідної інформації, результати трохи відстають у часі. Рівень прискорення оновлює рівень обслуговування, надсилаючи йому додаткові оновлення, враховуючи останні дані [8].

1.2. Огляд способів конструювання трафіку в SDN

1.2.1. Задача конструювання трафіку в SDN

Конструювання трафіку – це можливість керування напрямом проходження трафіку з метою досягнення певної цілі (резервування каналів, розподілення навантаження мережі, балансування і запобігання перевантаженню). Дане визначення дозволяє розділити поняття конструювання трафіку на складові і розглянути детальніше, які проблеми вирішує цей процес.

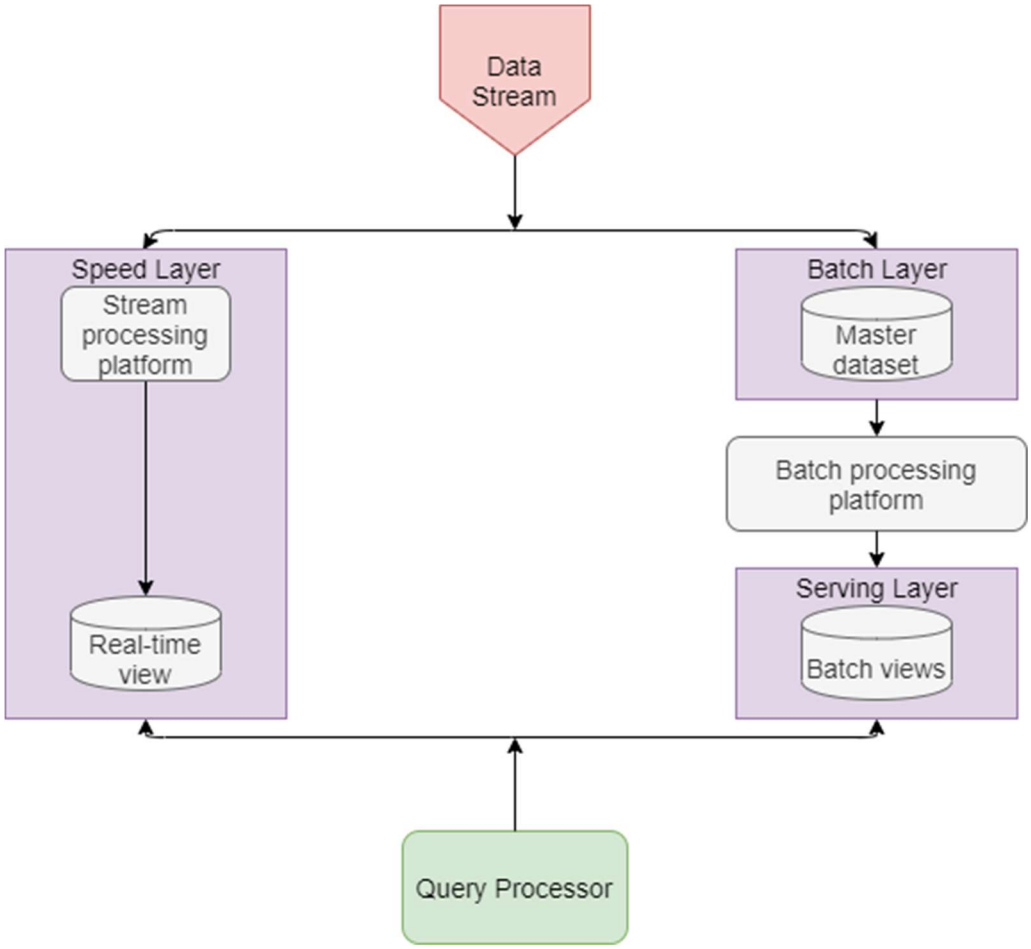


Рис. 1.6. Лямбда-архітектура

Підвищення продуктивності роботи мережі на рівні трафіка і ресурсів є основним завданням конструювання трафіка. Це досягається шляхом виконання орієнтованих на трафік вимог до продуктивності при забезпеченні економічного і надійного використання мережевих ресурсів. Орієнтовані на трафік характеристики містять в собі затримку і її варіації, частоту втрати пакетів і пропускну здатність. Важливою задачею конструювання трафіку є забезпечення надійного функціонування мережі. Надійність роботи мережі може бути досягнута за рахунок забезпечення механізмів, підвищуючи рівень цілісності мережі, а також її стійкість. Це призводить до мінімізації відмов у мережі, пов’язаних з помилками, збоями і пошкодженнями, що виникають в інфраструктурі. З цього слідує, що найбільш ваговим завданням є продуктивність, що доступна для кінцевих користувачів мережевого сервісу. На це потрібно зважати під час розробки механізмів і правил організації

трафіку. Характеристики, видимі кінцевим користувачам, визначаються новими властивостями мережі, які властиві мережі в цілому [19]

Конструювання трафіку має важливе значення в оптимізації продуктивності мережі шляхом аналізу трафіку в реальному часі і розробки механізмів маршрутизації для покращення використання мережевих ресурсів [20]. Використання SDN в мережі забезпечує більш ефективний спосіб виконання конструювання трафіку і підвищення продуктивності мережі.

При керуванні трафіком в програмно-конфігурованих мережах можна використати їх характеристики для вирішення задач, таких як керування потоками, відмовостійкість, оновлення топології, а також аналіз і характеристику трафіку.

Надійність мережі заснована на можливості швидкого відновлення у випадку збоїв в мережевих компонентах (контролерах, комутаторах, каналах зв'язку) [21]. Швидко відновитись після збою досить складно, так як контролер має розрахувати нові маршрути і передати їх усім працюючим комутаторам. Окрім цього, необхідно враховувати обмежені ресурси таблиці потоків на комутаторах, оскільки нові записи будуть додані в таблиці потоків працюючих комутаторів.

Функція оновлення топології керує запланованими змінами, такими як оновлення правил мережевої політики замість збою компонента. Контролер динамічно налаштовує нові правила глобальної політики. Під час виконання важливо підтримувати узгодженість правил політики для комутаторів, оскільки під час оновлення пакети можуть оброблятися різними правилами в різних комутаторах. Оновлення топології тим складніше, чим більша мережа, так як має бути виконана майже в режимі реального часу.

Аналіз трафіку грає особливо важливу роль у виявленні збоїв мережі чи каналу і прогнозування перенавантаження чи вузького місця. Інструменти моніторингу є основними елементами цієї важливої задачі. Для забезпечення робочої системи моніторингу необхідно забезпечення трьох аспектів [20]:

Технології проектування і моніторингу мережевих параметрів. Мережеві параметри – значення, що відображають поточний стан мережі. Вибір цих параметрів є попередньою умовою якісного керування мережею. Для моніторингу необхідні три типи параметрів:

- параметри топології мережі (кількість вузлів, пропускна здатність каналу, структура з'єднань і стан порта);
- параметри мережевого трафіка (кількість пакетів, що пересікають пристрої пересилки);
- параметри продуктивності мережі (показники для кожного потока: пропускна здатність, затримка, втрата пакетів тощо).

Загальна схема числення. Деякі системи числення SDN використовує традиційні методи мережевого трафіка IP, довільно обираючи локальні пакети для отримання статистики трафіку.

Аналіз і прогнозування трафіку. Ціллю є ідентифікувати аномальний трафік і проаналізувати можливе перенавантаження мережі. Надані дані допоможуть в покращенні планування і керування трафіком.

1.2.2. Огляд існуючих рішень для моніторингу та аналізу трафіку

Для традиційних IP-мереж доступні такі методи моніторингу, як NetFlow і sFlow. NetFlow, розроблений Cisco [22], надає статистику по окремих IP-потоках, підтримуючи кеш потоків, який відслідковує статистику для кожного з них. Він надає вичерпні дані про кожний потік (IP-адреса джерела і призначення, кількість байтів, кількість пакетів і номер порту). sFlow використовує ґрунтовану на часі вибірку для оцінки кількості пакетів і байтів у кожному потоці шляхом множення кількості вибіркових пакетів і байтів на коефіцієнт вибірки [23].

Багато інструментів для моніторингу трафіка були запропоновані для мереж OpenFlow. OpenNetMon [24] був розроблений для визначення факту, чи задовільняються потреби скрізного QoS і для того, що додатки конструювання трафіку могли обчислювати відповідні шляхи. Він реалізований як

додатковий модуль контролера SDN і опитує граничні комутатори для збору статистики потоку з адаптивною швидкістю, щоб визначити пропускну здатність, втрату пакетів і затримку.

SDN-Mon [25] був розроблений для моніторингу потоків і звільнення контролера від активності моніторингу, що дозволяє контролеру вставляти більш загальні правила пересилки в комутаторі. Містить два модулі: один на стороні контролера і інший на стороні комутатора. Перший модуль містить гнучкий моніторинг в контролері SDN. Другий модуль реалізований для керування функціями моніторингу на комутаторах.

У цьому джерелі [26] було запропоновано SDNMon для моніторинга рівню даних і покращення інформації про топологію на рівні керування. Ціль SDNMon - забезпечити пропускну здатність і затримку на двох рівнях деталізації: для кожного порта і кожного потока. Засновуючись на топології мережі, SDNMon використовує потоки для збору статистики портів і потоків і використовує два підходи до моніторингу: протокол sFlow і механізм опитування. Система дозволяє використовувати інші підходи через інтерфейс моніторинга.

Системи, наведені вище, надають ряд інструментів для моніторингу SDN, але вони базуються на алгоритмах для вирішення конкретних задач моніторинга і не мають визначення процеса збору, організації і обробки даних. Деякі основні види діяльності, такі як збереження зібраних даних для подальшої обробки і агрегація згенерованої статистики розробляються в залежності від їх прикладного значення. Також зазвичай надається одне конкретне значення, таке як пропускна здатність, затримка чи втрата пакетів. У роботі розглядається метод моніторингу мережевого трафіку на основі технологій Big Data, який встановлює процес збору даних лічильників, їх збереження, обробки для отримання статистики мережі і її збереження.

В системах аналізу трафіку виконується оцінка і прогнозування згенерованої матриці трафіку (TM) на основі процедур моделювання та

оптимізації. Для моделювання використовуються математичні моделі оцінки ТМ. Наприклад, iSTAMP [27] використовував стисле вимірювання агрегованого потоку і K найбільш інформативних потоків для генерації ТМ. Метод розбиває таблиці потоків на дві частини: перша для сукупних вимірів і друга для моніторинга окремих потоків. Використовуючи ці дві частини, iSTAMP оцінює матрицю трафіку. В даному джерелі [28] розглядається питання покращення оцінки матриці трафіку. Було розроблено спеціальний критерій, заснований на параметрі розподілення потоку, котрий дозволяє знайти підмножину трафіку, яка має бути виміряна, для покращення оцінки ТМ.

У роботі розглянуто підхід до проблеми оцінки ТМ з використанням методів обробки великих даних в реальному часі. Метод MapReduce використовує інфраструктуру прямого виміру для збору і опрацювання даних трафіку, що генерується OpenFlow. Даний підхід заснований на опитуванні портів комутаторів. Також пропонується з використанням розглянутого способу виконувати аналіз фізичного стану каналів та інтерфейсів для можливості швидкісного реагування на системні збої.

Висновки до розділу 1

У даному розділі було виконано огляд технологій SDN, наведені принципові відмінності програмно-конфігурованих мереж порівняно з традиційними IP-мережами. Обумовлено основні цілі та підходи конструювання трафіку і наведено огляд існуючих рішень для моніторингу та аналізу трафіку в SDN.

Зокрема, досліджено та проаналізовано основні принципи роботи технології SDN та її архітектури, а також протоколу OpenFlow. Головна відмінність програмно-конфігурованих мереж від традиційних полягає у можливій централізації і програмному керуванні всіма процесами, що проходять в мережі.

Розглянуто теоретичні та практичні аспекти роботи з великими даними, виділено базові принципи обробки даних в реальному часі. Для вирішення цієї задачі запропоновано застосовувати такі парадигми Big Data як MapReduce і лямбда-архітектуру, наведено особливості роботи вказаних технологій.

Також у даному розділі виконано огляд існуючих рішень для моніторингу та аналізу трафіку в SDN, що спираються на процедури моделювання та оптимізації. Спираючись на проведений аналіз, сформульовано задачі дослідження способу конструювання трафіку в програмно-конфігурованих мережах, що дозволятиме виконувати обробку даних в режимі реального часу. Досліджуваний спосіб матиме можливість швидкого реагування на зміни, що відбуваються у мережі, та їх прогнозування.

РОЗДІЛ 2. СПОСІБ КОНСТРУЮВАННЯ ТРАФІКУ В SDN НА ОСНОВІ ТЕХНОЛОГІЙ BIG DATA

2.1. Спосіб моніторингу трафіку на основі Big Data

Комутатор OpenFlow реалізовує підтримку лічильників, кожен з яких відображає певний трафік і котрі оновлюються з кожним новим пакетом. Спосіб моніторингу трафіку великих даних періодично опитує контролер SDN для отримання значень лічильників для ресурсів, які відслідковуються всередині комутатора. Лічильники підтримуються для декількох ресурсів, але для моніторингу будуть використані порти комутатора та таблиці потоків [91]. Основні значення, що надаються лічильниками комутаторів OpenFlow, можуть бути застосовані для забезпечення надання полоси пропускання мережі на рівні портів і потоків.

Спосіб моніторингу трафіку за допомогою Big Data містить в собі три основні процеси [29]:

1. *Збір даних.* Завантажує інвентарні дані мережі і трафіка з рівнів керування та інфраструктури за допомогою контролера SDN.
2. *Агрегація даних.* Обробляє отримані дані, обчислюючи параметри мережі в реальному часі.
3. *Акумуляція даних*. Зберігає і надає мережеві дані і параметри системам аналізу трафіка.

Збір даних. Операція опитує контролер для отримання необхідних даних для способу моніторингу трафіка. Зібрані дані можуть бути класифіковані як інвентарні дані і дані потокової передачі. Інвентарні дані заповнюють сховище, тоді як дані потокової передачі збираються і одразу передаються для подальшої обробки.

На рис. 2.1 [29] зображена модель взаємозв’язку сутностей для сховища інвентарних даних. Вони зберігаються в об’єктах Host, Interface, Switch Port, Switch, Flow Table, Flow.

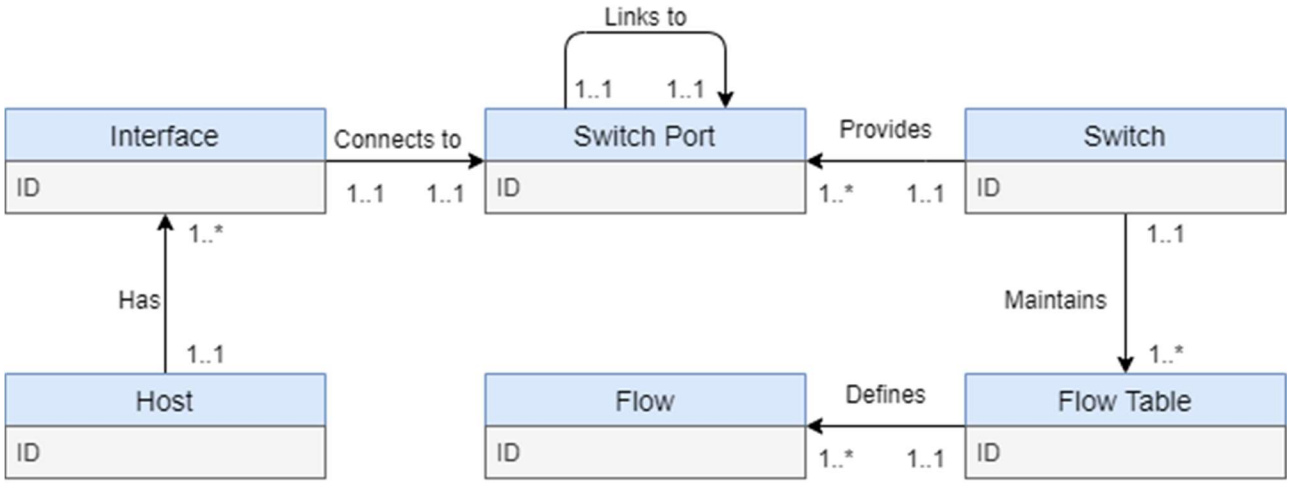


Рис. 2.1. ER-модель інвентарних даних

Потокові дані, пов'язані з лічильниками, що надаються пристроями OF. Так як лічильники оновлюються в кожному пакеті, що проходить через комутатор, швидкість зміни даних тут вища за швидкість зміни інвентарних даних. Тому при використанні цих лічильників для моніторингу чи іншого типу аналізу, їх значення мають періодично збиратися, а також після збору вони мають бути одразу надіслані для обробки.

Операція збору потокових даних (рис.2.2.) використовує північне API контролера для збору даних лічильників ресурсів, що відслідковуються, на кожному комутаторі. Ця дія складається з 6 задач, а саме: запуск збору лічильників, збір лічильників портів, збір лічильників таблиці потоків, збір лічильників потоків, форматування даних і відправка даних.

Наступні лічильники, які зберігаються на порту комутатора OpenFlow і які необхідні для реалізації способу моніторингу трафіка Big Data:

- Timestamp – момент, коли було здійснено читання;
- Packets Received – кількість отриманих пакетів;
- Packets Transmitted – кількість переданих пакетів;
- Bytes Received – кількість отриманих байт;
- Bytes Transmitted – кількість переданих байт;
- Collision Count – кількість зіткнень;

- Over RunError Received – кількість пакетів з переповненням RX;
- Drops Transmitted – кількість пакетів, відкинутих TX;
- Drops Received – кількість пакетів, відкинутих RX;
- Frame Error Received – кількість помилок вирівнювання фреймів;
- CRC Error Received – кількість помилок перевірки контрольної суми;
- Seconds – кількість секунд, протягом якого часу порт активний;
- Nanoseconds – кількість наносекунд в Seconds.

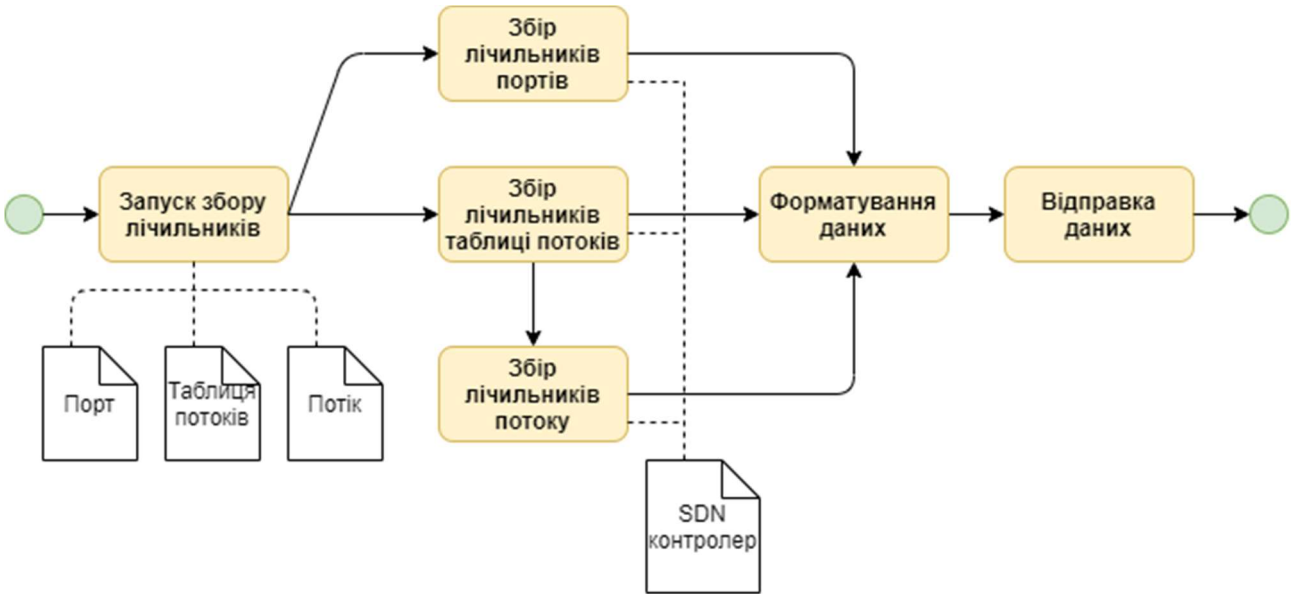


Рис. 2.2. Збір потокових даних

Агрегація даних. Цей процес агрегує дані, отримані під час збору даних, опрацьовує їх, генеруючи очікувану статистику, і надсилає результати на збереження. Процес, що виконується цією дією, містить підпроцеси черги повідомлень і генерації статистики.

Процес черги повідомлень складається з трьох базових етапів:

- *отримання повідомлення* – отримується повідомлення від процесу збору даних на порту, з таблиці потокі і потоку;

- *збереження повідомлення* – зберігаються повідомлення згідно заголовку (порт, таблиця потоку, потік), кожне повідомлення має бути збережене;
- *пересилка повідомлення* – відбувається пересилка повідомлення процесу, що надсилає запит.

Статистика періодично генерується на основі доступних даних повідомлень, збережених попереднім процесом. Ця задача надсилає запит на отримання даних, опрацьовує їх, обчислюючи очікувані параметри мережі і надсилає результат в сховище.

Акумуляція даних. У цьому процесі зберігаються інвентарні дані і згенеровані параметри мережі. Ці дані можуть використовуватись будь-якою системою моніторингу, аналізу трафіка або ж для візуалізації. Інвентарні дані можуть бути використані для відтворення топології мережі шляхом ідентифікації усіх її компонентів. Зібрана статистика шляхом під час виконання моніторингу може бути використана для генерації матриці трафіку. Для аналізу і прогнозування матриці трафіку.

2.2. Обчислення матриці трафіку за допомогою MapReduce

Підхід MapReduce складається з чотирьох функцій Map і однієї функції Reduce, як показано на рисунку 2.3.

Функція *Map Raw Data* отримує неопрацьовані дані у вигляді повідомлень і генерує мапу у вигляді ключ-значення, де ключем є часова мітка (timestamp) кожного повідомлення, а значенням – дані, необхідні для розрахунку пропускної здатності.

Функція *Sort By Key* сортує ключі, згенеровані функцією Map Raw Data, в порядку зростання. Цей крок необхідний для розрахунку пропускної здатності каналу. Поточна пропускна здатність каналу використовується для розрахунку пропускної здатності певного маршруту.

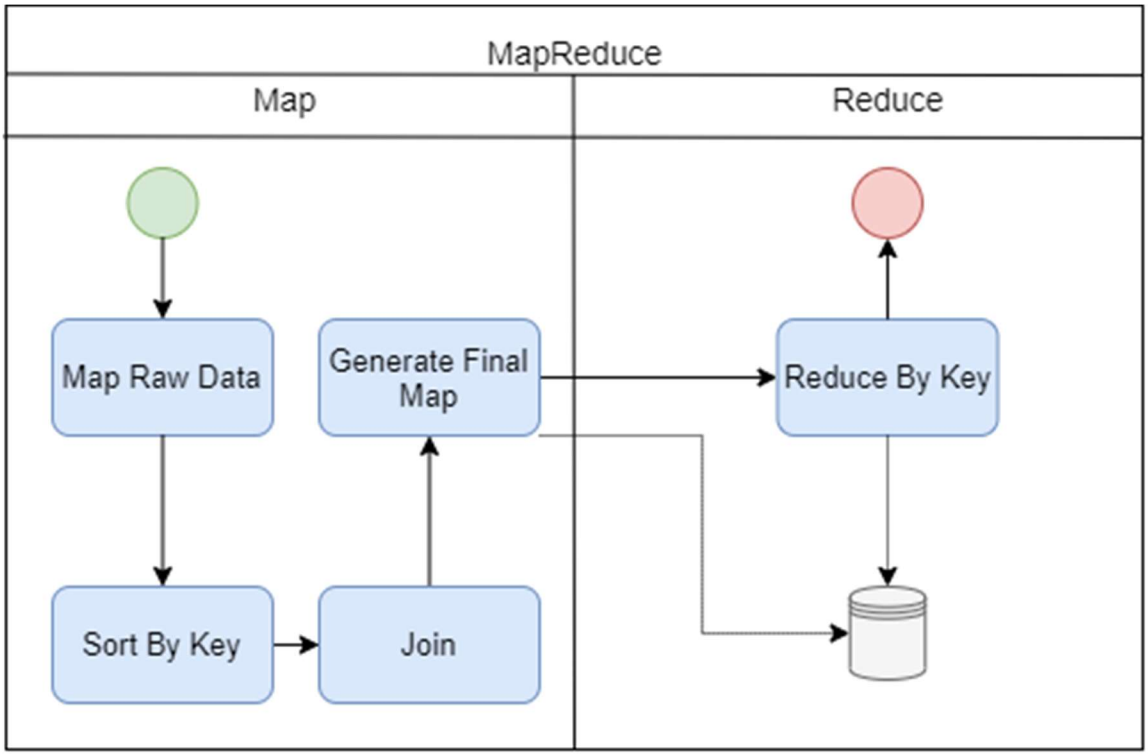


Рис. 2.3. Модель MapReduce

Функція *Generate Final Map* виконує дві задачі: обчислення поточної пропускної здатності в кожному каналі і обчислення накопиченої пропускної здатності в кожному каналі.

Функція *Reduce By Key* генерує і зберігає фінальну таблицю матриці трафіку і обробляє її.

Першочергова ціль підходу MapReduce – визначити і згенерувати ключі, які в подальшому будуть агреговані і, як наслідок, скорочені. Розглянутий підхід пропонує встановлювати, що ключі будуть складатися з усіх зв'язків на маршруті між парами стартовим хостом – хостом призначення (origin – destination, або OD). Для досягнення цієї мети мають бути ідентифіковані усі необхідні елементи, котрі будуть використовуватися для генерації ключів. Наведено нижче основні визначення елементів:

- Множина $N = \{x_1, x_2, \dots, x_n\}$ містить всі вузли мережі.
- Множина $H = \{h_1, h_2, \dots, h_m \mid h_i \in N\}$ містить всі хости мережі.

- Множина $S = \{s_1, s_2, \dots, s_k \mid s_i \in N\}$ містить всі комутатори в мережі. Кожен комутатор має певну кількість портів, що позначаються $s_{i1}, s_{i2}, \dots, s_{il}$.
- Множина $U_1 = \{(h_i, s_j) \vee (s_j, h_i) \mid h_i \in H \wedge s_j \in S\}$ визначає з'єднання між хостами і комутаторами.
- Множина $U_2 = \{(s_i, s_j) \mid s_i \in S \wedge s_j \in S \wedge i \neq j\}$ визначає з'єднання комутатор-комутатор.
- Множина $U = U_1 \cup U_2$.
- Мережа визначається графом $G(N, U)$.
- Множина $L_1 = \{(h_i, s_j, s_{ja}) \vee (s_j, h_i, s_{ja})\}$ визначає канали між хостами і портами комутаторів.
- Множина $L_2 = \{(s_i, s_{ia}, s_j, s_{jb}) \mid i \neq j\}$ визначає канали між двома комутаторами (їх портами).
- $L = L_1 \cup L_2$.
- $HC = \{(h_i, h_j) \mid (h_j, h_i) \in H \times H \wedge i \neq j\}$ визначає комбінацію усіх пар OD в мережі.
- Для шляху $h_i \longrightarrow h_j$ P=$(h_i, s_1), (s_1, s_2), (s_2, s_3), \dots, (s_{k-1}, s_k), (s_k, h_j)$.
- $LP = \{(s_{ij}, (h_1, h_2), \dots, (h_n, h_m) | i = \overline{1..k}, j = \overline{1..l}\}$. Це множина $s_{ij}s_{ij}$.

Для генерації матриці трафіку необхідне виконання певних умов:

1. Маємо множини *H, S,* і граф *G.*
2. Побудована множина *HC* використовуючи функцію $f: H \longrightarrow HC$. Величина НС задається комбінацією усіх елементів *H(n)* по два елементи.
3. Отримана множина *L.*

Множина *LP* містить для кожного порту комутатора список хостів, для яких цей порт є частиною маршруту. Для побудови цієї множини, спершу

необхідно згенерувати множину *HC,* що містить комбінації усіх пар OD хостів. Для кожної пари в *HC* знаходиться шлях *P*. Кожен шлях між членами пари OD складається з трьох типів з'єднань: між стартовим хостом і першим комутатором; між комутаторами; між останнім комутатором і хостом призначення. Для кожного типу пари виконується пошук множини L, щоб знайти порти, використані в маршруті. Для з'єднаня першого і третього типу для пошуку використовуємо множину L_1, і повертаємо порт s_{ij}. Для з'єднань другого типу використовуємо множину L_2, повертаємо пару портів (s_{ia}, s_{jb}). Для кожного порта, що повернули, додаємо пару *(port, (O, D))* і додаємо в множину *LP*.

Дані, доступні з процесу черги повідомлень, описаного в попередньому підрозділі, містять інформацію про трафік в мережі. Ці дані в сукупності з елементами множини *LP* використовуються для генерації ключів частини Map процесу MapReduce.

Функція *Map Raw Data* отримує статистику трафіка у вигляді повідомлень (рис. 2.4.). Поле часової мітки надається у форматі Unix, де останні три цифри означають мілісекунди часу збору даних. Пропонується відкинути ці цифри для отримання пропускної здатності в секундах. Це спрощення також дозволяє групувати статистику, зібрану за одну секунду.

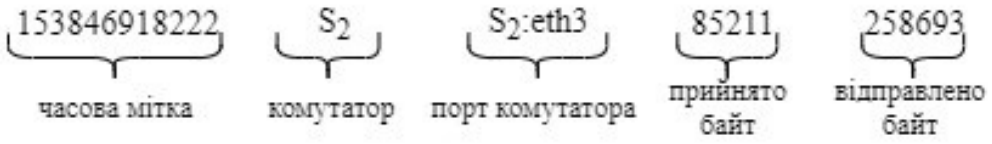


Рис. 2.4. Модель повідомлення статистики порту

Map, отриманий за допомогою *Map Raw Data* сортується функцією *Sort By Key*, де ключ є полем часової мітки і порту комутатора.

Функція *Generate Final Map* являє собою два алгоритми, необхідні для генерації кінцевої таблиці: алгоритм *Generate LP Set*, що генерує множину *LP*, і *Generate (key,value) pairs*, який для кожного повідомлення статистики порту

обчислює пропускну здатність каналу, який включає в себе цей порт і генерує ключ і значення для усіх пар OD, що включають цей порт.

2.3. Аналіз трафіку на основі матриці трафіку

Розглядається метод розширеного аналізу трафіка у вигляді діаграми діяльності. Потік керування, представлений на діаграмі дій, встановлюється на основі залежності даних між діями для забезпечення різних рівнів агрегації (рис. 2.5.).

Операція *Store Streaming Data* отримує неопрацьовані дані, що поступають від контролера SDN, які забезпечують вихідну точку для статистичних обчислень. Процес *Calculate Switch Port Traffic* обчислює базовий рівень статистики трафіка порту. Статистика трафіка порту дозволяє генерувати декілька рівнів агрегації, таких як трафік комутатора, мережевий трафік, трафік каналу і трафік маршруту. Операція *Generate Switch Traffic Statistics* агрегує статистику комутації по часових мітках. Статистика комутатора містить агрегацію трафіка в портах комутатора. Процес *Generate Network Switch Statistics* і *Generate Network Port Statistics* використовують одне й те саме джерело даних, вони можуть працювати паралельно, оскільки вони виводять різні рівні агрегації. Операція *Generate Switch Traffic Statistics* об’єднує мережеву статистику на основі статистики комутатора.

Значення, генеровані до цього моменту, представляють статистику, обчислену для комутаторів, портів і мережі, і позволяють генерувати аналіз трафіку. Операція *Generate Switch Traffic Analysis* забезпечує аналіз трафіку окремих комутаторів. *Generate Switch Port Traffic Analysis* реалізує кожного порту кожного комутатора. Оскільки пара портів комутатора створює канал зв’язку, процес *Generate Link Traffic Analysis,* що генерує аналіз трафіку каналу, залежить від результатів роботи попереднього процесу. Такий самий рівень залежності застосовується і до *Generate Path Traffic Analysis,* оскільки шляхи між хостами складаються з каналів, і, відповідно, необхідно спочатку проаналізувати трафік кожного каналу.

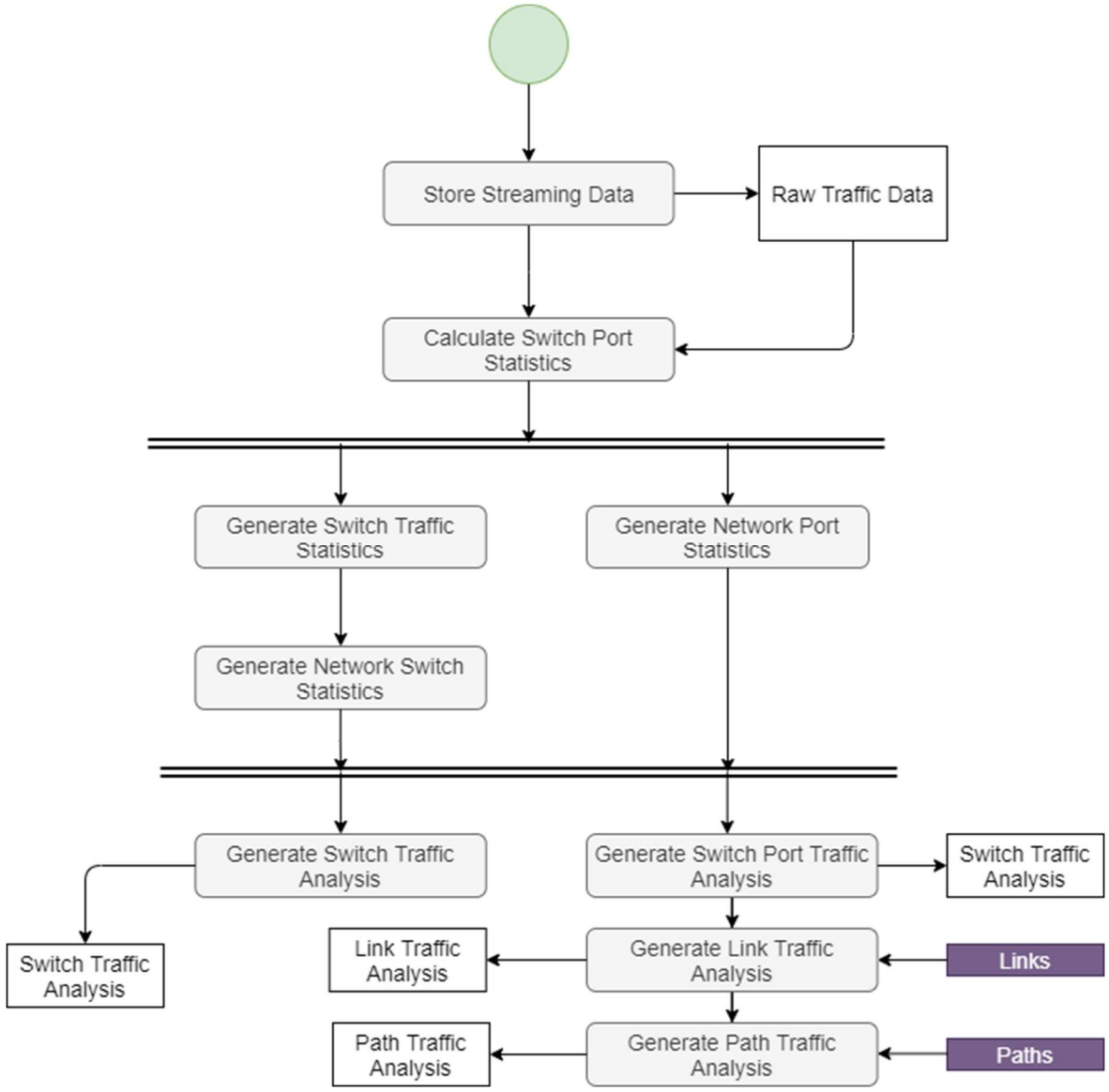


Рис. 2.5. Метод аналізу трафіку

Операція *Store Streaming Data* збирає неопрацьовані дані трафіка, що надходять від контролера SDN і зберігає їх в об'єкті *Raw Traffic Data.* Основний набір даних – це фізичне сховище процесу *Raw Traffic Data*. Пакетна задача опрацьовує весь вміст основного набору даних. Таким чином, ця операція постійно додає нові дані і ніколи не видаляє старі. Також поточний процес форматує вхідні дані для подальшої обробки.

На відміну від об'єкту повідомлення, зображеного на рис. 2.4., макет зібраних неопрацьованих даних також містить в собі значення прийнятих пакетів (PR) і відправлених пакетів (PT). Вказані значення в сукупності з

отриманими і переданими байтами (BR і BT) є усіма необхідними лічильниками кожного порту комутатора.

Таблиця 2.1 відображає опис обчислення статистики, яку генерує процес *Calculate Switch Port Statistics* для кожного порту. Усі інші наступні дії використовують дані, згенеровані цим процесом, для виконання певних рівнів агрегації

Операція *Generate Switch Traffic Analysis* виконує операції MapReduce для генерації статистики комутатора. Згенерована статистика необхідна для обчислення трафіку комутатора і виконання аналізу трафіка порту комутатора. Ключ, за яким збираємо статистику має вигляд *(switch, timestamp)*.

Таблиця 2.1. Розрахована пропускна здатність для порту

Розрахована пропускна здатність	**Визначення**
Packet Received Throughput (PRT)	$\frac{PR}{S}$ (2.1)
Packet Transmitted Throughput (PTT)	$\frac{PT}{S}$ (2.2)
Packet Throughput (PTr)	$\frac{(PT+PR)}{S}$ (2.3)
Bytes Received Throughput (BRT)	$\frac{BR}{S}$ (2.4)
Bytes Transmitted Throughput (BTT)	$\frac{BT}{S}$ (2.5)
Bytes Throughput (BTr)	$\frac{(BT+BR)}{S}$ (2.6)

За допомогою вищенаведеного ключа генерується наступна статистика:

Підсумовування. Кожен елемент таблиці 2.1 має бути підсумований. Наприклад, $TPRT = \sum_{p=1}^{n} PRT$ підсумовує значення пропускної здатності комутатора по отриманих пакетах, $TPTT = \sum_{p=1}^{n} PTT$ підсумовує значення

пропускної здатності комутатора по відправлених пакетах, і так само для решти лічильників.

Середнє значення. Обчислюємо середнє значення для кожної величини для комутатора, використовуючи формулу для середнього арифметичного:

$$M = \frac{1}{n} * \sum_{i=1}^{n} p_i, \quad (2.7)$$

де M – середнє значення, n – кількість портів у комутаторі, p – значення лічильника на певному порту i.

Середньоквадратичне відхилення. Відхилення для кожного елемента обчислюється за формулою:

$$\sigma = \sqrt{\frac{\sum(x-M)^2}{n}}, \quad (2.8)$$

де σ – середньоквадратичне відхилення, x – значення лічильника на певному порту комутатора, M – середнє значення лічильника для комутатора, обчислене за формулою 2.7, n – кількість різних значень.

Статистика, згенерована під час процесу Generate Network Switch Statistics, є сукупністю значень, отриманих в результаті попередніх дій. Використання агрегованої по комутатору статистики дозволяє розглянутому методу забезпечувати аналіз трафіка комутатора на основі статистики мережевого трафіка. Ця дія генерує статистику того ж типу, що і попередня, з тією лише різницею, що значення відносяться до усієї мережі. Як ключ, що використовується для агрегації даних, використовуємо поле *(timestamp).*

Окрім генерації статистики мережі комутаторів, використовуючи ключ *(timestamp)*, можливо згенерувати статистику на основі портів, яка необхідна для генерації аналізу трафіку портів комутаторів. Основною різницею з попередньою активністю є обчислення середнього значення – не зважаючи на те, що сума статистики по усіх портах мережі і комутаторах буде однаковою, середні значення по портах і середньоквадратичне відхилення є різним. Тому розглянутий спосіб пропонує обидві дії для генерації мережевої статистики.

Generate Switch Traffic Analysis. Ця активність вводить z-оцінку аналізу трафіку комутатора. Z-оцінка відображає, скільки і які значення відрізняються від середнього, враховуючи середньоквардатичне відхилення. Додатня z-оцінка означає, що значення вище за середнє, а негативна z-оцінка значить, що значення нижче за середнє.

Значення з показниками менше за -3 або більше за 3 називається *викидом.* Поведінка трафіка в будь-якому комутаторі може бути проаналізована шляхом обчислення z-оцінки наступних параметрів: отримані пакети (PR), надіслані пакети (PT), отримані байти (BT), надіслані байти (BR), пропускна здатність по отриманих пакетах (PTR), пропускна здатність по надісланих пакетах (PTT), загальна пропускна здатність по пакетах (PTr), пропускна здатність по отриманих байтах (BTR), пропускна здатність по надісланих байтах (BTT), загальна пропускна здатність по байтах (BTr),

Z-оцінка обчислюється за наступною формулою:

$$z = \frac{x-\mu}{\sigma}, \qquad (2.9)$$

де z – z-оцінка, x – значення певного лічильника, μ – середнє по певному значенню, σ – середньоквадратичне відхилення по певному лічильнику.

Об’єкт *Switch Traffic Analysis* зберігає значення, згенеровані в поточному процесі, а також в процесах генерації статистики комутаторів і мережі. Для будь-якого додатку, котрому необхідно отримати статистичне значення і історичну поведінку кожного комутатора в мережі, об’єкт аналізу трафіку комутатора є джерелом даних. Кожен запис, окрім статистики комутатора, також містить статистику мережі.

Незважаючи на те, що такий підхід потребує більше місця для зберігання, будь-який аналізатор виграє від цього, тому що за допомогою лише однієї операції читання може отримувати статистичні дані про комутатори і мережу, значно пришвидшивши процес аналізу.

Процес *Generate Switch Port Traffic Analysis* генерує z-оцінку портів комутатора. Розраховуються показники кожного порту в контексті комутатора і в контексті мережі. За допомогою операції *Join* дозволяє обчислити z-оцінку для усіх необхідних параметрів.

Кожен канал мережі складається з двох портів, що належить різним комутаторам і не підключений до хоста, в іншому випадку канал – порт комутатора. Об'єкт *Links* – контейнер для каналів, сформованих в мережі. Обирається канал з максимальним значенням $BR + BT$. Операція *Join* виконується між результатом виконання процесу *Generate Switch Port Traffic Analysis* і об'єктом *Links.* Операція обирає максимальне значення $BR + BT$ кожного посилання в кожній часовій мітці і зберігає результат в об'єкті *Link Traffic Analysis.*

Ціль процесу *Generate Path Traffic Analysis* є аналіз трафіка всіх пар OD хостів. У цій операції використовується об'єкт *Paths,* котрий утримує набір усіх з'єднань між парами OD хостів. Операція *Join* виконується між вихідними даними процесу *Generate Link Traffic Analysis* і об'єктом *Paths*, щоб заповнити таблицю результатів. В таблиці 2.2 наведено приклад оформлення результатів виконання процесу.

Таблиця 2.2. Приклад об'єкту аналізу трафіка маршруту

Timestamp	**Path**	**Path calculated statistic**	**Path traffic analysis**	**Path statistic**
14382672	S1:h1-h8	S1:h1-h8 statistics columns	path analysis	path statistic
14382672	S1:h2-h8	S1:h2-h8 statistics columns	path analysis	path statistic
14382672	S1:h3-h7	S1:h3-h7 statistics columns	path analysis	path statistic
14382672	S2:h4-h8	S2:h4-h8 statistics columns	path analysis	path statistic
14382672	S2:h4-h5	S2:h4-h5 statistics columns	path analysis	path statistic
14382672	S3:h1-h2	S3:h1-h2 statistics columns	path analysis	path statistic
14382672	S1:h2-h6	S1:h2-h6 statistics columns	path analysis	path statistic
… … …				

Висновки до розділу 2

У цьому розділі розглянуто спосіб конструювання трафіка, використовуючи сучасні методи й парадигми Big Data, який враховує особливості роботи SDN-мереж, і, зокрема, роботи протоколу OpenFlow і комутаторів OpenFlow. Описано основні принципи роботи систем моніторингу і аналізу трафіку для подальшої програмної реалізації імітаційної моделі.

Описано спосіб моніторингу трафіку, що використовує парадигми Big Data. Наведено основні етапи для системи моніторингу: збір, агрегація і акумуляція даних. В якості даних використовуватимуться лічильники портів комутаторів OpenFlow. Формується черга повідомлень для подальшого збереження і обробки.

Розглянуто спосіб обчислення і прогнозування матриці трафіку з використанням парадигми MapReduce. Описані основні об’єкти, що використовуються для генерації матриці трафіку. Наведено два необхідні для реалізації алгоритми для обчислення і прогнозування, а саме генерація множини *LP* і генерація фінальної мапи ключ-значення.

Аналіз трафіку пропонується виконувати на основі отриманої матриції трафіку. Наведена діаграма процесів аналізу трафіку і їх детальний опис. В результаті, процес збору даних лічильників портів комутатоів OpenFlow дозволяє згенерувати аналіз трафіку і пропускної здатності кожного маршруту OD у мережі

Окрім цього, пропонується використовувати лічильники помилок на портах комутаторів для виконання аналізу фізичного стану каналів і подальшого використання зібраної інформації при конструюванні трафіку.

РОЗДІЛ 3. ПРОЕКТУВАННЯ І РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1. Топологія мережі і опис засобів реалізації

Для реалізації способу конструювання трафіку в програмно-конфігурованих мережах на основі парадигм Big Data була змодельована певна мережа, топологія якої складається з восьми хостів, шести комутаторів OpenFlow і одного контролера (рис. 3.1).

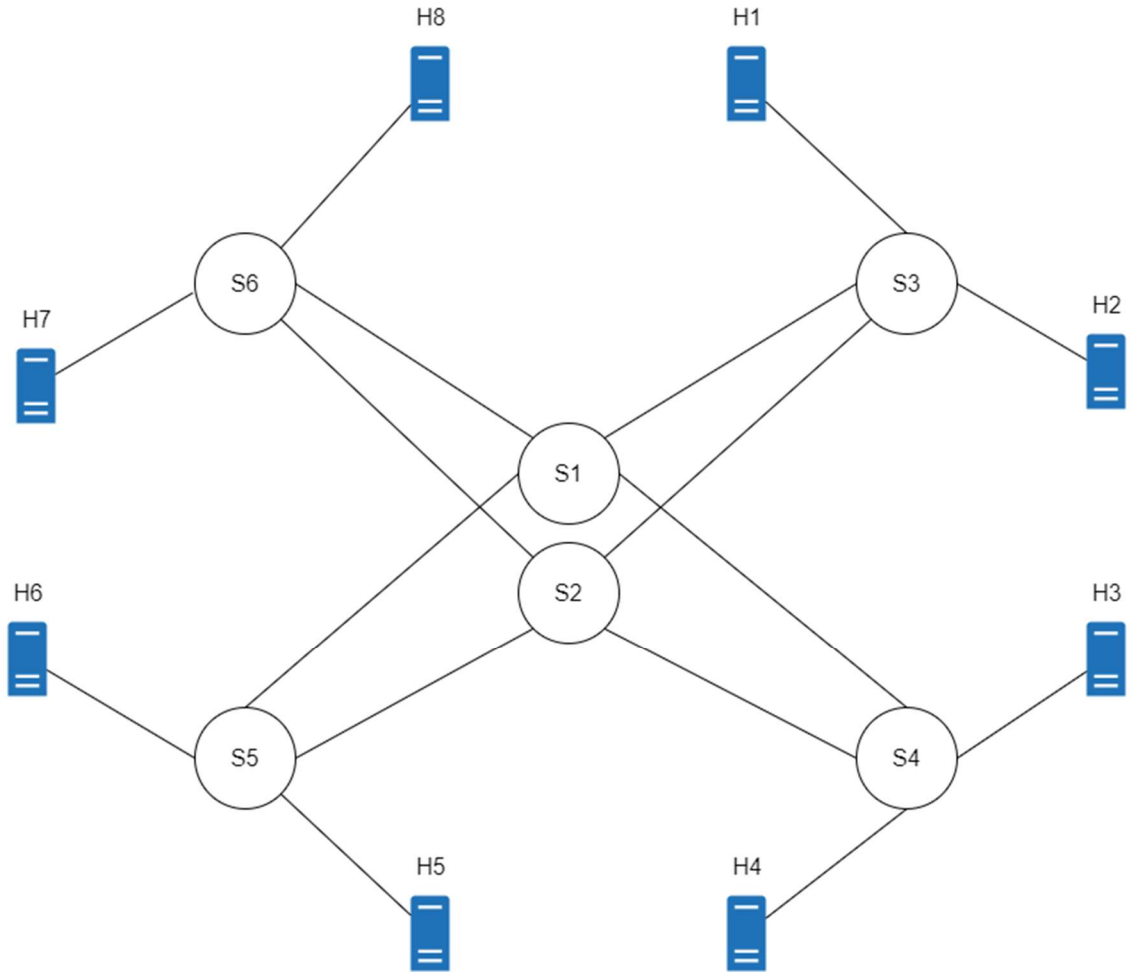


Рис. 3.1. Топологія модельованої мережі

Для моделювання використано мову Python, зокрема середовище для моделювання мережі Mininet та бібліотеку для роботи з графами networkx. Вибір обумовлений можливістю створення довільної топології та масштабування моделі, широкими можливостями Python для проведення необхідних обчислень, швидкодією і зручністю роботи з даними. Як

модельоване сховище даних було використано систему управління базами даних SQLite. Окрім цього, мова Python є передвстановленою на більшості сучасних контролерів SDN, що дозволяє в подальшому реалізувати повноцінний модуль системи моніторингу та аналізу трафіку безпосередньо на контролері.

3.2. Опис логіки програми

Для реалізації системи моніторингу за допомогою середовища Mininet була змодельована мережа, зображена на рис. 3.1.. Конструювання трафіка починається з генерації певного трафіка, який дозволив зібратися лічильниками на портах комутаторів OpenFlow та в таблицях потоків. Контролер з періодичністю *interval* опитує комутатори з метою збору лічильників, згідно з процесом *Збору даних,* описаному в підрозділі 2.1.. Значення *interval* залежить від завантаженості процесору контролера і його здатності до опрацювання інформації. У створеній моделі це значення дорівнює трьом секундам.

Зібрані дані передаються до процесу *Агрегації даних,* що створює чергу повідомлень, необхідних до опрацювання, а також зберігає статистичні дані в сховищі сирих даних. Об’єкт повідомлення реалізований класом *Message* і має наступні атрибути:

- часова мітка (timestamp);
- порт комутатора (switchport);
- комутатор (switch);
- пакетів отримано (PR);
- пакетів передано (PT);
- байт отримано (BR);
- байт передано (BT);
- секунди, протягом яких порт є активним (sec);
- активність в наносекундах (nanosec).

Зібрані об’єкти повідомлень, відповідно до процесу *Акумуляції даних* зберігається в об’єктах *Черги повідомлень*, готових до опрацювання.

Опрацювання *черги повідомлень* згідно з парадигмою Big Data MapReduce і подальше отримання детальної інформації по маршрутах має на меті послідовне виконання двох функцій: *Generate LP set* і *Generate Final Map*, умови для виконання яких описані в підрозділі 2.2..

Генерація множини *LP* виконується за наступним алгоритмом, наведеним на рис.3.2.. У модельованій мережі для створення цієї множини були визначені множини *H, S, L, P,* які задаються інформацією про топологію модельованої мережі.

Функція *getPath()* приймає як параметри пару OD і повертає поточний маршрут для цієї пари. Наприклад, для пари OD (h_2, h_5) функція має повернути маршрут: (h_2, s_3, s_1, s_5, h_5), а для маршруту (h_1, h_8) результатом буде (h_1, s_3, s_2, s_6, h_8).

Функція *getPorts()* приймає як параметри канал визначеного маршруту і повертає порти, які є задіяними в даному каналі, тобто встановлює відповідність між точкою маршруту і портом.

Результатом роботи алгоритму є словник, у якому ключем є певний порт комутатора OpenFlow, а значенням є певний маршрут у вигляді (кореневий комутатор / стартовий хост / хост призначення), наприклад, для порту s4:eth4 список усіх маршрутів:

- S2 / h1 / h3;
- S2 / h2 / h3;
- S2 / h3/ h5;
- S2 / h3 / h6;
- S2 / h3 / h7;
- S2 / h3 / h8;
- S2 / h1 / h4;
- S2 / h2 / h4;

- S2 / h4/ h5;
- S2 / h4 / h6;
- S2 / h4 / h7;
- S2 / h4 / h8.

На основі даних про маршрути, отриманих після виконання функції *Generate LP set* потрібно виконати задачу генерації фінальної мапи, що містить в собі дані про трафік кожного каналу в мережі. У цьому випадку ключем є пара OD, а дані про трафік є значенням. Алгоритм цієї функції зображений в Додатку 3.

Для кожного об'єкту повідомлення статистики порту вилучається добувається повідомлення пари портів з однаковою часовою міткою. Порт і його пара є елементами множини *L.* Якщо порт є елементом підможини L_1, то порт і його пара мають одне й те саме значення, так як зв'язок формується між портом хоста і портом комутатора, але порт хоста не контролюється. Для елементів підмножини L_2 знаходить відповідна пара і, якщо поля секунд однакові для обох портів, значення BR для вихідного порта і BT для його пари мають бути також однаковими. Однак збір статистики для обох портів в одну й ту саму секунду і наносекунду є нереальним сценарієм, отже, актуальну статистику містить одне з цих повідомлень. Для того, щоб використати саме актуальну статистику, використовується повідомлення з найбільшим значенням BR + BT, по якій буде обчислена пропускна здатність каналу.

Фінальний етапом алгоритму є генерація мапи. Надходить запит до множини LP для надання шляхів через обраний порт. Для кожного маршруту, що повертається, генерується шукана пара.

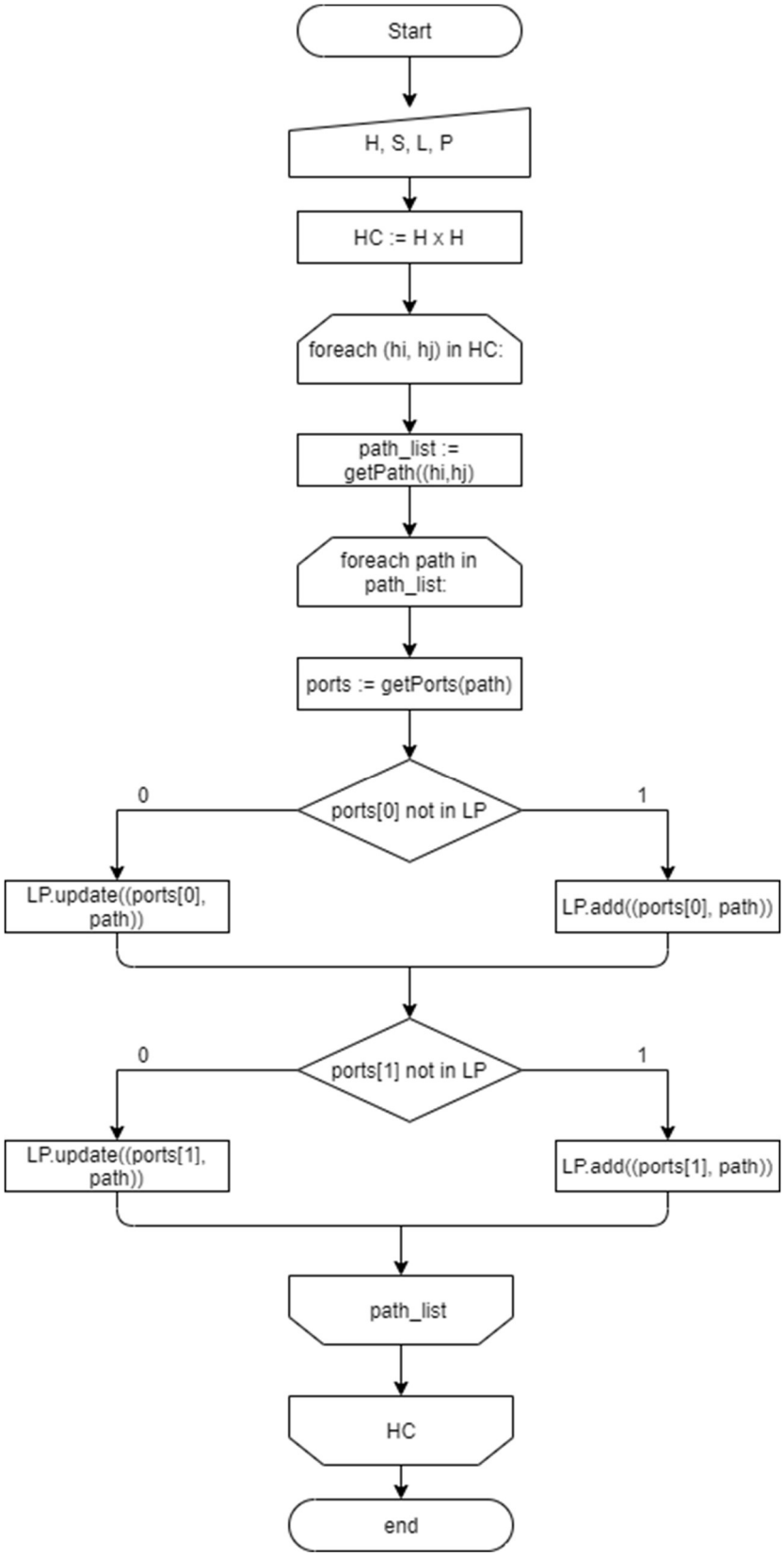


Рис. 3.2. Алгоритм генерації LP-множини

Іншим аспектом конструювання трафіку є його аналіз. Статистику з акумульованих повідомлень можна використовувати в даних цілях. При зборі

лічильників також додаються поля отриманих і переданих пакетів (PR і PT). Згідно з рис. 2.5, зібрана статистика використовується для генерації статистики трафіку портів комутаторів, яка, в свою чергу, в подальшому використовується для генерації аналізу трафіку маршруту.

Для збереження отриманого аналізу створюються таблиці бази даних, що відповідають кожному пункту проведення аналізу. Доступ до зчитування та редагування даних реалізовано через спеціальну бібліотеку sqlite3.

Як видно з рис. 2.5, є можливість роботи з таблицями паралельно – наприклад, після генерації статистики порту комутатора можна паралельно генерувати як мережеву статистику по портах, так і статистику по кожному комутатору. Цей нюанс є дуже важливим для швидкодії системи аналізу, а також при подальшому масштабуванні топології, і саме паралельне опрацювання даних є однією з головних особливостей розглянутого методу конструювання трафіку на основі Big Data.

Кожну зі створених і опрацьованих таблиць можна використовувати для конструювання трафіку, візуалізації, аналізу поведінки мережі тощо. Статистичні дані у них є актуальними, що дозволяє їх використання у сучасних системах моніторингу та аналізу трафіку, а також можливе використання їх у контексті передбачення трафіку – у дійсно великих системах зі значним обсягом даних є можливість застосування даних для створення математичних моделей, здатних з високою точністю передбачити пропускну здатність за певний час.

3.3. Інструкція користувача

Реалізація програмного інтерфейсу для системи конструювання трафіку виконана на мові програмування Python. Програмний продукт складається з декількох функціональних модулів та бази даних:

- клас Message;
- модуль Generate Key_Value Pairs;
- модуль Analytics;

- модуль Plotter;
- база даних Analytics.

При побудові інтерфейсу було враховано наступний функціонал:

1. Можливість редагувати мережу, додавати / видаляти нові комутатори і хости і змінювати зв’язки між ними.
2. Збереження графу мережі.
3. Вивід статистичних даних у вигляді візулацізації.
4. Можливість налаштовувати систему для конкретних задач моніторингу та аналізу трафіку.

Клас Message відповідає за генерацію повідомлень зі статистичною інформацією лічильників портів комутаторів OpenFlow з вищеописаними параметрами.

Модуль Generate Key_Value Pairs дозволяє задати мережу і отримати множину *LP* і згенерувати матрицю трафіку для подальшої оцінки, використовуючи чергу повідомлень *messages,* що складається з об’єктів класу Message.

Модуль Analytics реалізує зв’язок з базою даних Analytics і, використовуючи чергу повідомлень *messages* і згенеровані попереднім модулем Generate Key_Value Pars множину LP і матрицю трафіку виконує аналіз трафіку і акумуляцію статистичних даних, вносячи отримані результати у відповідні таблиці бази даних. Лістинг коду наведено в Додатку 4.

Модуль Plotter дозволяє отримати статистичні результати у вигляді графіків, використовувати їх в інших системах моніторингу або для демонстрації роботи реалізованої системи.

Структурна та функціональна схема розроблюваного програмного продукту наведені в Додатку 1 та Додатку 2 віповідно.

Висновки до розділу 3

У даному розділі було виконано розгляд засобів, використаних для створення моделі системи для конструювання трафіку на основі технологій Big Data, зазначена топологія модельованої мереж. Описано логіку роботи розробленого програмного продукту, а також наведена інструкція користувача для використання.

Зокрема, обґрунтовано вибір мови програмування та бази даних для роботи програмного продукту. Наведено граф мережі, що описує топологію модельованої SDN, на основі якої буде продемонстрована робота програмного продукту.

Виділено основні принципи роботи реалізованої програми. Зазначено нюанси роботи певних модулів, особливості обраних шляхів обробки даних та створення статистики. Описані основні алгоритми системи моніторингу та генерації матриці трафіку, а також принципі роботи з базою статистичних даних.

Також наведена інструкція користувача: описано існуючі модулі, способи використання створеного програмного продукту. Зазначені необхідні залежності для коректної роботи програми.

РОЗДІЛ 4. МОДЕЛЮВАННЯ І ПРЕДСТАВЛЕННЯ РЕЗУЛЬТАТІВ

4.1. Моделювання

Проведемо моделювання в розробленому програмному продукті з метою отримання результатів роботи системи конструювання трафіку в SDN на основі технологій Big Data. На основі топології мережі, зображеної на рис. 3.1., визначимо основні множини H, S, L. З цих множин отримаємо множину всіх можливих пар OD та маршрутів P.

Згідно з алгоритмом генерації LP-множини (рис. 3.2.), для кожного порта комутатора OpenFlow отримаємо список маршрутів, що через нього проходять (табл. 4.1.).

Таблиця 4.1. Загальний вигляд LP-множини

Порт	**Маршрути**							
s1:eth1	h1->h3	h1->h4	h1->h5	h1->h6	h1->h7	h1->h8	...	h2->h8
s1:eth2	h1->h3	h1->h4	h2->h3	h2->h4	h3->h5	h3->h6	...	h4->h8
s1:eth3	h1->h5	h1->h6	h2->h5	h2->h6	h3->h5	h3->h6	...	h6->h8
s1:eth4	h1->h7	h1->h8	h2->h7	h2->h8	h3->h7	h3->h8	...	h6->h8
s2:eth1	h1->h3	h1->h4	h1->h5	h1->h6	h1->h7	h1->h8	...	h2->h8
s2:eth2	h1->h3	h1->h4	h2->h3	h2->h4	h3->h5	h3->h6	...	h4->h8
s2:eth3	h1->h5	h1->h6	h2->h5	h2->h6	h3->h5	h3->h6	...	h6->h8
s2:eth4	h1->h7	h1->h8	h2->h7	h2->h8	h3->h7	h3->h8	...	h6->h8
s3:eth1	h1->h2	h1->h3	h1->h4	h1->h5	h1->h6	h1->h7	...	h1->h8
s3:eth2	h1->h2	h2->h3	h2->h4	h2->h5	h2->h6	h2->h7	...	h2->h8
… … ...								
s6:eth4	h1->h7	h1->h8	h2->h7	h2->h8	h3->h7	h3->h8	...	h6->h8

Тепер, коли множину LP згенеровано, можна розпочинати генерацію мапи ключ / значення. Для генерації використовуються попередньо зібрана

черга повідомлень зі значеннями лічильниками портів комутаторів OpenFlow (рис 2.1), інтервал, та множина LP.

Одним з основних полів повідомлення є поле timestamp, яке ідентифікує повідомлення по шкалі часу і дозволяє хронологічно структурувати отриману статистику. Це поле використовуємо як частину ключа. Іншою частиною ключа є опис маршруту у вигляді *кореневий комутатор / стартовий хост / хост призначення*.

Як значення словників використовується статистика пропускної здатності, яка обчислюється для кожного каналу згідно з даними, що знаходяться в певному повідомленні.

Результат виконання функції мапування зображений на таблиці 4.2.

Таблиця 4.2. Результат виконання функції мапування

Ключ	**Значення** (байт за інтервал)
s3/h1/h5 1522598234301	(768730854.375, 1950526742.667)
s1/h3/h8 1522598234301	(883367641.833, 1697735390.333)
s2/h2/h6 1522598234301	(703538326.444, 2017217664.667)
s3/h1/h2 1522598234301	(83656294.714, 118236094.333)
s4/h4/h5 1522598234301	(73039966.167, 90373040.333)
s3/h1/h2 1522598234301	(68608148.429, 74676766.667)
s1/h2/h7 1522598234301	(521281163.222, 1448446833.667)
s2/h5/h7 1522598234301	(713604204.714, 1596548049.333)
… … …	
s6/h5/h8 1522598234301	(71426788.857, 90861917.333)

Як бачимо з таблиці 4.2, отримані дані можуть містити однакові ключі. Таке явище обумовлено наявністю даних з різними значеннями, тому необхідно виконати операцію Reduce By Key і згрупувати їх за ключами для отримання фінальної мапи. Результат представлений у таблиці 4.3.

Таблиця 4.3. Матриця трафіку

Ключ	**Значення** (байт за інтервал)
s2/h1/h5 1522598234301	(1745912250.309, 3565277454.333)
s2/h1/h6 1522598234301	(1745912250.309, 3565277454.333)
s2/h3/h6 1522598234301	(1857912600.976, 3791546156.0)
s2/h3/h5 1522598234301	(1857912600.976, 3791546156.0)
s6/h6/h7 1522598234301	(74435925.0, 76716585.0)
s4/h1/h3 1522598234301	(58372997.666, 97222631.333)
s4/h2/h3 1522598234301	(58372997.666, 97222631.333)
s3/h1/h6 1522598234301	(68525913.0, 95686057.666)
s3/h1/h3 1522598234301	(68525913.0, 95686057.666)
s6/h1/h8 1522598234301	(41315784.222, 66058336.666)
s6/h6/h8 1522598234301	(41315784.222, 66058336.666)
s3/h2/h8 1522598234301	(67284260.14285715, 85525815.0)
s5/h3/h5 1522598234301	(61566526.888, 81854078.333)
s5/h5/h6 1522598234301	(141599148.31746033, 185315823.3333333)
… … …	
s5/h3/h6 1522598234301	(80032621.42857143, 103461745.0)

Для обраної топології результуючий словник має сто зібраних значень статистики на кореневих комутатор для певного маршрута у визначений момент часу. Згенерована матриця трафіку шляхом MapReduce може бути використана для досконалішого конструювання трафіку.

4.2. Опис результатів

За обраною топологією виконаємо процес аналізу трафіку. Маємо чергу повідомень з різними часовими мітками і доданими параметрами отриманих і переданих пакетів (PR і PT). Відбувається послідовне

опрацювання повідомлень з подальшим додаванням статистичних даних то відповідних таблиць статистики, згідно з процесом, зображеним на рис. 2.5.

На рисунках 4.1, 4.2, 4.3, та 4.4 наведені витримки з таблиць бази даних, заповнених за визначеними алгоритмами, дані у яких обчислені за попередньо наведеними формулами (2.1. – 2.6.).

Часова мітка	Порт	Комутатор	PR	PT	BR	BT	PRT	PTT	PTr	BRT	BTT	BTr
1522598234298	s3:eth2	s3	816405	1061909	81639833	106192180	272135	353970	626105	35397393	27213278	62610671
1522598234298	s1:eth4	s1	50038268	76014844	1250957363	1900370582	12509567	19003711	31513278	475092646	312739341	787831986
1522598234298	s2:eth4	s2	48313036	66992228	1207825902	1674805576	9662607	13398446	23061053	334961115	241565180	576526296
1522598234298	s3:eth3	s3	1899006	1337783	189899275	133779931	379801	267557	647358	26755986	37979855	64735841
1522598234298	s4:eth4	s4	1850303	1518279	185028577	151829999	616768	506093	1122861	50610000	61676192	112286192
1522598234298	s5:eth4	s5	862703	1017768	86268956	101778634	287568	339256	626824	33926211	28756319	62682530
1522598234298	s3:eth1	s3	1055806	1888502	105580064	188849176	211161	377700	588862	37769835	21116013	58885848
1522598234298	s6:eth1	s6	1444943	1448313	144493060	144832993	288989	289663	578651	28966599	28898612	57865211
1522598234298	s1:eth1	s1	46723924	66493004	1168097463	1662324183	11680981	16623251	28304232	415581046	292024366	707605412
1522598234298	s6:eth4	s6	1240066	1387181	124008516	138717232	248013	277436	525449	27743446	24801703	52545150

Рис. 4.1. Згенерована статистика портів комутаторів

Часова мітка	Комутатор	PR	PT	BR	BT	PRT	PTT	PTr	BRT	BTT	BTr
1522598234298	s3	4916986	6088912	491694689	608892401	1245020	1599466	2844486	159946919	124500984	284447904
1522598234301	s3	37106334	34643092	558887016	409412788	14795755	14851483	29647238	172951163	219450368	392401531
1522598234304	s3	20064328	25999648	501609908	649986882	7981605	10384586	18366191	259612973	199540676	459153649
1522598234307	s3	25876788	28050224	646915982	701257497	9435726	10044294	19480020	251108430	235891839	487000269
1522598234310	s3	21494520	22105056	537365614	552621593	7220252	7824173	15044425	195602945	180507769	376110713
1522598234313	s3	25971436	20709588	649286570	517740830	11133859	8577661	19711521	214442367	278346400	492788768
1522598234316	s3	25149708	19771144	628741360	494282254	7940138	6862665	14802803	171567840	198502502	370070342
1522598234319	s3	20913480	20205744	522841495	505141275	9813075	9533581	19346656	238338377	245329166	483667543
1522598234322	s3	22297620	21513824	557441405	537846689	8277033	7470274	15747307	186756918	206926082	393683000
1522598234325	s3	22901160	23437372	572521869	585934195	7456608	8066302	15522910	201657437	186412876	388070313
1522598234328	s3	18774048	26606900	469356480	665169367	6962472	10543706	17506178	263591056	174064310	437655366
1522598234331	s3	26289040	26321792	657226471	658051731	10721853	9953201	20675054	248832498	268045851	516878350
1522598234334	s3	20946884	24538152	523666752	613447204	6661474	8197184	14858658	204927553	166535000	371462552
1522598234337	s3	23060768	19114048	576521125	477854562	7880064	5944463	13824527	148612885	197001929	345614814
1522598234340	s3	23382508	21112716	584566031	527818822	9092472	7165378	16257850	179134573	227313861	406448434
1522598234343	s3	20738396	18263692	518461744	456584398	7079793	6179567	13259360	154486793	176995120	331481913
1522598234346	s3	21245912	24504908	531147657	612623031	7742288	8944466	16686754	223611908	193557223	417169130
1522598234349	s3	19779132	16654836	494473080	416371768	7891203	6823327	14714529	170583634	197277628	367861262
1522598234352	s3	23581384	27755320	589534673	693887517	7399063	8627935	16026998	215699858	184976621	400676479

Рис 4.2. Згенерована статистика комутаторів

Окрім обчислених даних пропускної здатності по пакетах та байтах, важливо зазначити, що, починаючи з таблиці статистики по комутаторах, також формуються значення середнього арифметичного та середньоквадратичного відхилення по відповідних значеннях лічильників. Вони необхідні для подальшого надання z-оцінки пропускної здатності шляху, та зберігаються у відповідних таблицях.

Часова мітка	PR	PT	BR	BT	PRT	PTT	PTr	BRT	BTT	BTr
1522598234298	430998507	526976757	12274775199	14904073575	97745125	120057995	217803119	3455267286	2839349762	6294617048
1522598234301	603776889	584119043	13594605166	12873320720	213879315	200921266	414800581	4344571458	4773297125	9117868582
1522598234304	567221976	477151468	14180547813	11928778863	194140286	166239722	360380008	4155990005	4853505551	9009495556
1522598234307	535382560	590920460	13384556103	14773021262	176191893	186112238	362304132	4652808717	4404794775	9057603491
1522598234310	582994916	574624424	14574889197	14365611407	217504783	214671476	432176260	5366786362	5437625383	10804411745
1522598234313	600771320	567839112	15019278463	14195980356	193134560	182998638	376133198	4574967654	4828361462	9403329116
1522598234316	588006676	467255572	14700175519	11681390471	235115683	192334777	427450460	4808368455	5877895464	10686263919
1522598234319	555896600	515717036	13897414936	12892921822	194781385	189899188	384680573	4747477790	4869534317	9617012107
1522598234322	491391148	620534412	12284777572	15513358923	182907568	231245068	414152636	5781126292	4572689706	10353815998
1522598234325	537600412	512075284	13440002436	12801886333	196082204	180906067	376988270	4522654038	4902052394	9424706432
1522598234328	582311908	553492284	14557804911	13837308012	213002261	201118009	414120270	5027949903	5325059079	10353008982
1522598234331	518351528	516441604	12958784906	12911039373	182463300	181127040	363590339	4528174122	4561581313	9089755435
1522598234334	553677568	559537552	13841936120	13988437097	222532395	218039882	440572277	5450995853	5563308728	11014304581
1522598234337	514424568	561327216	12860613337	14033185115	176974813	192301544	369276357	4807540468	4424369290	9231909758
1522598234340	540430024	529749944	13510764082	13243744271	195953236	182794437	378747673	4569859155	4898837191	9468696346
1522598234343	485824716	544595792	12145622876	13614881956	181616688	211410606	393027293	5285261890	4540418844	9825680734
1522598234346	454892528	516203444	11372301349	12905091062	149079345	163703145	312782491	4092581580	3726980409	7819561989
1522598234349	525466756	512098112	13136666867	12802454923	223108591	213223969	436332560	5330601513	5577714016	10908315529
1522598234352	510688172	570408372	12767203705	14260216569	176817616	205310680	382128296	5132770175	4420441257	9553211433

Рис. 4.3. Згенерована мережева статистика по комутаторах

Часова мітка	PR	PT	BR	BT	PRT	PTT	PTr	BRT	BTT	BTr
1522598234298	430998507	526976757	12274775199	14904073575	97745125	120057995	217803119	3455267286	2839349762	6294617048
1522598234301	603776889	584119043	13594605166	12873320720	213879315	200921266	414800581	4344571458	4773297125	9117868582
1522598234304	567221976	477151468	14180547813	11928778863	194140286	166239722	360380008	4155990005	4853505551	9009495556
1522598234307	535382560	590920460	13384556103	14773021262	176191893	186112238	362304132	4652808717	4404794775	9057603491
1522598234310	582994916	574624424	14574889197	14365611407	217504783	214671476	432176260	5366786362	5437625383	10804411745
1522598234313	600771320	567839112	15019278463	14195980356	193134560	182998638	376133198	4574967654	4828361462	9403329116
1522598234316	588006676	467255572	14700175519	11681390471	235115683	192334777	427450460	4808368455	5877895464	10686263919
1522598234319	555896600	515717036	13897414936	12892921822	194781385	189899188	384680573	4747477790	4869534317	9617012107
1522598234322	491391148	620534412	12284777572	15513358923	182907568	231245068	414152636	5781126292	4572689706	10353815998
1522598234325	537600412	512075284	13440002436	12801886333	196082204	180906067	376988270	4522654038	4902052394	9424706432
1522598234328	582311908	553492284	14557804911	13837308012	213002261	201118009	414120270	5027949903	5325059079	10353008982
1522598234331	518351528	516441604	12958784906	12911039373	182463300	181127040	363590339	4528174122	4561581313	9089755435
1522598234334	553677568	559537552	13841936120	13988437097	222532395	218039882	440572277	5450995853	5563308728	11014304581
1522598234337	514424568	561327216	12860613337	14033185115	176974813	192301544	369276357	4807540468	4424369290	9231909758
1522598234340	540430024	529749944	13510764082	13243744271	195953236	182794437	378747673	4569859155	4898837191	9468696346
1522598234343	485824716	544595792	12145622876	13614881956	181616688	211410606	393027293	5285261890	4540418844	9825680734
1522598234346	454892528	516203444	11372301349	12905091062	149079345	163703145	312782491	4092581580	3726980409	7819561989
1522598234349	525466756	512098112	13136666867	12802454923	223108591	213223969	436332560	5330601513	5577714016	10908315529
1522598234352	510688172	570408372	12767203705	14260216569	176817616	205310680	382128296	5132770175	4420441257	9553211433

Рис. 4.4. Згенерована мережева статистика по портах

Також можна помітити, що дані мережевої статистики по комутатору і по портах є однаковими - сумарний трафік, що пройшов через усі порти і через усі комутатори за певні часові відрізки є однаковим. Основна різниця полягає саме в обчисленні середнього арифметичного та середньоквадратичного відхилення, що пов’язано з проходженням різної кількості трафіку через конкретний порт і конкретний комутатор. Як приклад, детальніше розглянемо дані отриманих пакетів для часової мітки 1522598234298 (таблиця 4.4):

Таблиця 4.4. Порівняння середнього та відхилення для статистики по портах і комутаторах

Величина	**Статистика по комутаторах**	**Статистика по портах**
Пакетів отримано (PR)	430998507	430998507
Пропускна здатність (Tr)	217803119.43	217803119.43
Середнє MeanPR	71833084.5	17958271.125
Середнє MeanTr	36300519.9	9075129.97
Відхилення DeviationPR	94520776.85	24216509.72
Відхилення DeviationTr	47437712.48	12248501.06

Згідно з методами, застосованими для отримання статистики, використовуємо статистику портів та мережеву статистику по портах сукупно з множинами каналів та шляхів для отримання аналізу трафіку у вигляді z-оцінок пропускної здатності і статистичних значень відповідно до певних часових міток. Частина результатів, відсортована за найбільшою пропускною здатністю, наведена на рис. 4.5. Відповідні Z-оцінки наведені на рис. 4.6.

Як видно з рис. 4.6, маршрути, кореневим комутатором яких є s1, потребують в ремашрутизації - z-оцінка пропускної здатності більша за 3.

Отже, отримані дані дозволяють провести аналіз і виконати необхідне конструювання трафіка для отримання оптимальних значень мережі.

Наочне застосування отриманих даних можна передати за допомогою візуалізації і графіків. Значення об’єктів, для яких необхідно отримати візуалізовані дані, отримуються від користувача за допомогою спеціального

інтерфейсу (рис. 4.7). На рисунках 4.8, 4.9, 4.10 представлені дані пропускної здатності порту, комутатора та певного маршруту.

Машрут	Часова мітка	PR	PT	BR	BT	PRT	PTT	PTr	BRT	BTT	BTr
s2/h5/h2	1522598234553	76463440	78402768	1911587468	1960065951	38231720	39201384	77433104	980032976	955793734	1935826710
s2/h5/h1	1522598234553	76463440	78402768	1911587468	1960065951	38231720	39201384	77433104	980032976	955793734	1935826710
s2/h6/h2	1522598234553	76463440	78402768	1911587468	1960065951	38231720	39201384	77433104	980032976	955793734	1935826710
s2/h2/h6	1522598234553	76463440	78402768	1911587468	1960065951	38231720	39201384	77433104	980032976	955793734	1935826710
s2/h1/h5	1522598234553	76463440	78402768	1911587468	1960065951	38231720	39201384	77433104	980032976	955793734	1935826710
s2/h6/h1	1522598234553	76463440	78402768	1911587468	1960065951	38231720	39201384	77433104	980032976	955793734	1935826710
s2/h1/h6	1522598234553	76463440	78402768	1911587468	1960065951	38231720	39201384	77433104	980032976	955793734	1935826710
s2/h2/h5	1522598234553	76463440	78402768	1911587468	1960065951	38231720	39201384	77433104	980032976	955793734	1935826710
s1/h3/h2	1522598234544	74080972	79666068	1852023635	1991653988	37040486	39833034	76873520	995826994	926011818	1921838812
s1/h1/h6	1522598234544	74080972	79666068	1852023635	1991653988	37040486	39833034	76873520	995826994	926011818	1921838812
s1/h4/h1	1522598234544	74080972	79666068	1852023635	1991653988	37040486	39833034	76873520	995826994	926011818	1921838812
s1/h4/h2	1522598234544	74080972	79666068	1852023635	1991653988	37040486	39833034	76873520	995826994	926011818	1921838812
s1/h3/h1	1522598234544	74080972	79666068	1852023635	1991653988	37040486	39833034	76873520	995826994	926011818	1921838812
s1/h6/h1	1522598234544	74080972	79666068	1852023635	1991653988	37040486	39833034	76873520	995826994	926011818	1921838812
s1/h1/h3	1522598234544	74080972	79666068	1852023635	1991653988	37040486	39833034	76873520	995826994	926011818	1921838812
s1/h1/h5	1522598234544	74080972	79666068	1852023635	1991653988	37040486	39833034	76873520	995826994	926011818	1921838812
s1/h2/h3	1522598234544	74080972	79666068	1852023635	1991653988	37040486	39833034	76873520	995826994	926011818	1921838812
s1/h6/h2	1522598234544	74080972	79666068	1852023635	1991653988	37040486	39833034	76873520	995826994	926011818	1921838812
s1/h1/h4	1522598234544	74080972	79666068	1852023635	1991653988	37040486	39833034	76873520	995826994	926011818	1921838812

Рис. 4.5. Генерація аналізу трафіку маршрутів

s2/h5/h2	1522598234553	1,86665	1,83537	1,86665	1,83536	2,62071	2,60243	2,64300	2,60243	2,62071	2,64299
s2/h5/h1	1522598234553	1,86665	1,83537	1,86665	1,83536	2,62071	2,60243	2,64300	2,60243	2,62071	2,64299
s2/h6/h2	1522598234553	1,86665	1,83537	1,86665	1,83536	2,62071	2,60243	2,64300	2,60243	2,62071	2,64299
s2/h2/h6	1522598234553	1,86665	1,83537	1,86665	1,83536	2,62071	2,60243	2,64300	2,60243	2,62071	2,64299
s2/h1/h5	1522598234553	1,86665	1,83537	1,86665	1,83536	2,62071	2,60243	2,64300	2,60243	2,62071	2,64299
s2/h6/h1	1522598234553	1,86665	1,83537	1,86665	1,83536	2,62071	2,60243	2,64300	2,60243	2,62071	2,64299
s2/h1/h6	1522598234553	1,86665	1,83537	1,86665	1,83536	2,62071	2,60243	2,64300	2,60243	2,62071	2,64299
s2/h2/h5	1522598234553	1,86665	1,83537	1,86665	1,83536	2,62071	2,60243	2,64300	2,60243	2,62071	2,64299
s1/h3/h2	1522598234544	2,08488	2,23436	2,08488	2,23436	3,15916	3,11064	3,15994	3,11064	3,15915	3,15994
s1/h1/h6	1522598234544	2,08488	2,23436	2,08488	2,23436	3,15916	3,11064	3,15994	3,11064	3,15915	3,15994
s1/h4/h1	1522598234544	2,08488	2,23436	2,08488	2,23436	3,15916	3,11064	3,15994	3,11064	3,15915	3,15994
s1/h4/h2	1522598234544	2,08488	2,23436	2,08488	2,23436	3,15916	3,11064	3,15994	3,11064	3,15915	3,15994
s1/h3/h1	1522598234544	2,08488	2,23436	2,08488	2,23436	3,15916	3,11064	3,15994	3,11064	3,15915	3,15994
s1/h6/h1	1522598234544	2,08488	2,23436	2,08488	2,23436	3,15916	3,11064	3,15994	3,11064	3,15915	3,15994
s1/h1/h3	1522598234544	2,08488	2,23436	2,08488	2,23436	3,15916	3,11064	3,15994	3,11064	3,15915	3,15994
s1/h1/h5	1522598234544	2,08488	2,23436	2,08488	2,23436	3,15916	3,11064	3,15994	3,11064	3,15915	3,15994
s1/h2/h3	1522598234544	2,08488	2,23436	2,08488	2,23436	3,15916	3,11064	3,15994	3,11064	3,15915	3,15994
s1/h6/h2	1522598234544	2,08488	2,23436	2,08488	2,23436	3,15916	3,11064	3,15994	3,11064	3,15915	3,15994
s1/h1/h4	1522598234544	2,08488	2,23436	2,08488	2,23436	3,15916	3,11064	3,15994	3,11064	3,15915	3,15994

Рис. 4.6. Z-оцінки для маршрутів

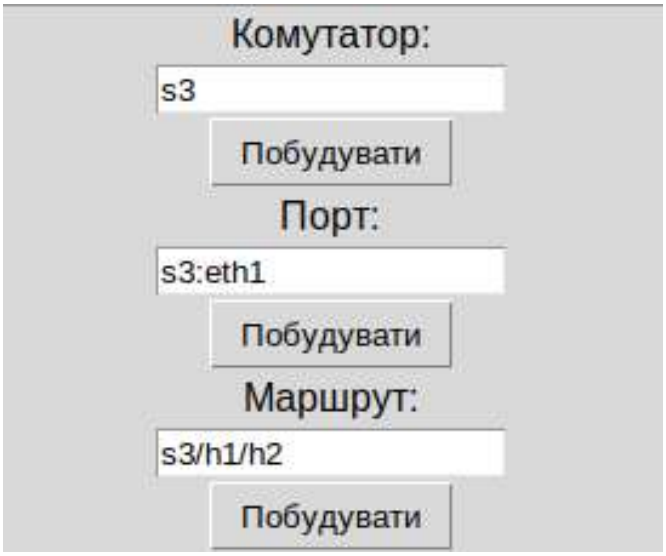


Рис. 4.7. Інтерфейс для отримання візуалізації

Рис. 4.8. Пропускна здатність комутатора S3

Рис. 4.9. Пропускна здатність порта S3:eth1

Рис. 4.10. Пропускна здатність по маршруту h1/h2 через комутатор S3

По візуалізованих даних досить просто зрозуміти характер, що носить трафік: чи є статистика вірною, чи відбувся викид даних. Таким чином, якщо порівняти статистичну інформацію з таблиці 4.5 та графіку з рис. 4.10., то можна зрозуміти, що насправді отримане значення z-оцінки є швидше викидом, аніж перевантаженням маршруту. Розуміння характеру трафіку дозволяє зосередитись на відповідних процесах ремаршрутизації.

Висновки до розділу 4

У цьому розділі було виконано моделювання системи конструювання трафіку в програмно-конфігурованих мережах з використанням технологій Big Data і аналіз отриманих результатів з метою здобуття статистичної інформації мережі, генерування історичної інформації, а також створено і проаналізовано матрицю трафіку модельованої мережі.

Зокрема, описано програмний продукт, що реалізує систему моніторингу та аналізу трафіку у визначеній мережі SDN. Обґрунтовано вибір використаних технологій. Наведено результати роботи визначених алгоритмів генерації LP-множини та створення матриці трафіку з використанням парадигми MapReduce.

Продемонстровано результати роботи програмного модулю для отримання статистичної і історичної інформації по мережі з метою використання у конструюванні трафіку. Виконано графічну візуалізацію отриманих даних, а також закцентовано увагу на нюансах роботи з розробленим програмним продуктом.

Наведені результати підтверджують коректність роботи розробленої програми та засвідчують працездатність і актуальність обраного методу конструювання трафіку в SDN.

ВИСНОВКИ

У даній дипломні роботі досліджено і реалізовано метод конструювання трафіку в програмно-конфігурованих мережах, що базується на використанні парадигм Big Data. Розроблено програмний продукт, що моделює систему моніторингу та аналізу трафіку з використанням технології MapReduce. У рамках роботи були визначені і виконані наступні завдання:

1. Проаналізовано архітектуру SDN і роботу протоколу OpenFlow, досліджено існуючі методи конструювання трафіку та недоліки їх використання.
2. Обрано актуальний метод конструювання трафіку в SDN на основі техноголій Big Data, що дозволяє паралельно опрацьовувати великі обсяги інформації майже в реальному часі.
3. Проаналізовано метод моніторингу трафіку, що вирішує проблему дрібнозернистого моніторингу та дозволяє виконувати оперативне балансування трафіку в мережу.
4. Досліджено метод аналізу трафіку, що заснований на використанні актуальних даних і історичної інформації.
5. Розроблено програмний продукт, здатний моделювати мережу SDN з певною топологією і проводити моніторинг і аналіз трафіку з метою подальшого використання для конструювання.

Обидві досліджені активності використовують однакові дані, але направлені на вирішення різних цілей. Реалізований метод дозволяє зосередитись на типі моніторингу замість вирішення задач об’єднання даних на різних рівнях ієрархії мережі.

Тема конструювання трафіку в SDN знаходиться на ранніх етапах дослідження. Досі розробляються нові і покращуються існуючі рішення, тож апрям подальших досліджень може бути обраний у бік реалізації подібної системи з використанням лямбда-архітектури, висвітлення недоліків і переваг кожного з методів. Розглядаючи інші вектори дослідження, можна обрати

напрям динамічного оновлення топології мережі, заснованого на статистичних та історичних даних, зібраних з використанням технологій Big Data.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

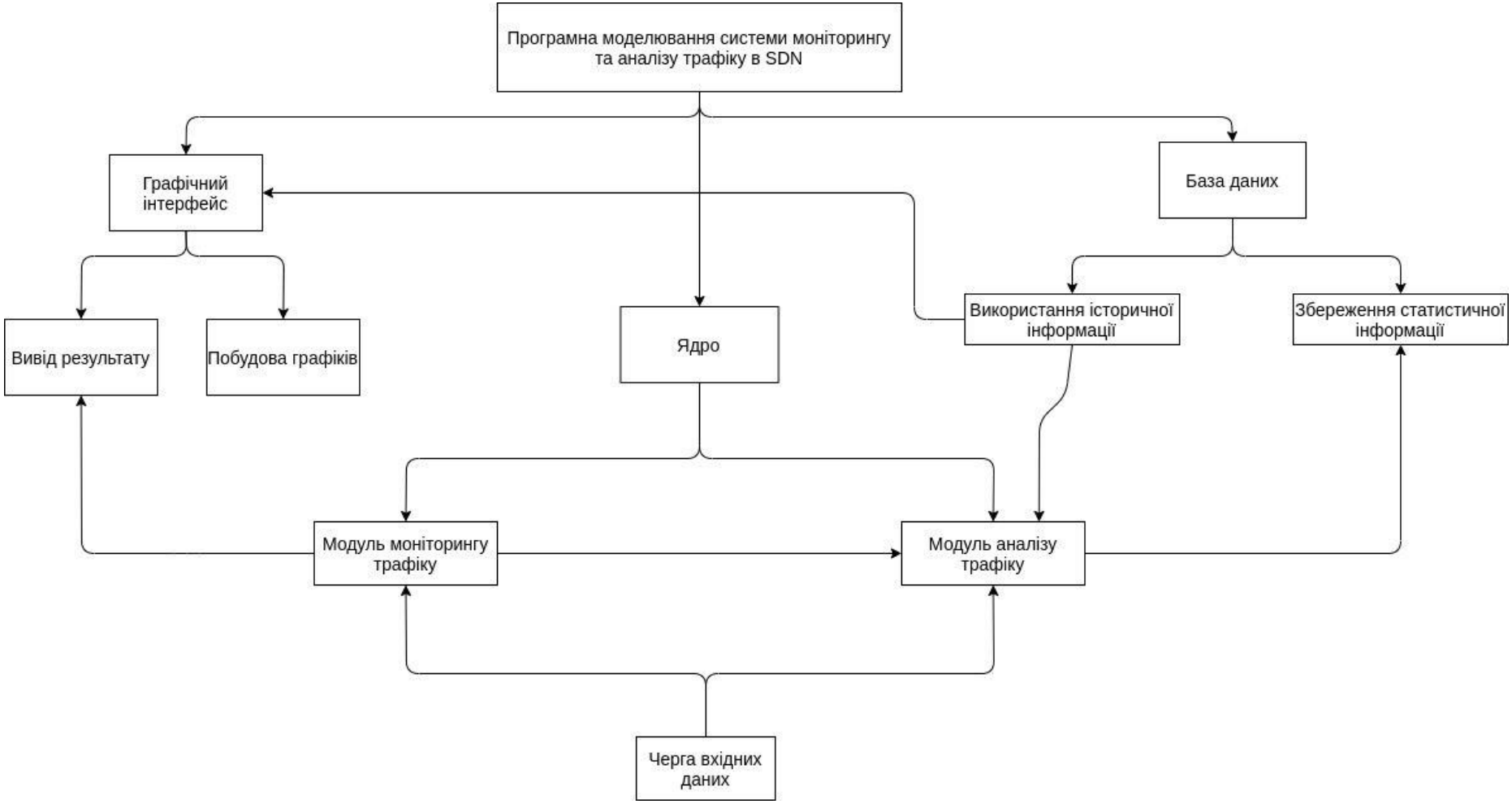
1. H. Farhady, H. Lee, and A. Nakao, “Software-Defined Networking: A survey,” Computer Networks, vol. 81, , 2015.
2. D. Kreutz, F. M. V. Ramos, P. E. Ver´ısssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-Defined Networking : A Comprehensive Survey,” Proceedings of the IEEE, vol. 103, 2015.
3. Software-Defined Networking (SDN) Definition [Електронний ресурс] // Open Networking Foundation – Режим доступу до ресурсу: https://www.opennetworking.org/sdn-definition/
4. I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, “A roadmap for traffic engineering in software defined networks,” Computer Networks, vol. 71, 2014.
5. Концепция и преимущества программно-определяемых сетей SDN [Електронний ресурс] // Infocom, 2016. – Режим доступу до ресурсу: http://infocom.uz/2016/10/27/koncepciya-i-preimushhestva-programmno-opredelyaemyx-setej-sdn/
6. W. Stallings, “Software-Defined Networks and OpenFlow,” The Internet Protocol Journal, vol. 16, 2013.
7. SDN Architecture Overview [Електронний ресурс] // Open Networking Foundation – Режим доступу до ресурсу: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf
8. A. Hakiri, A. Gokhale, P. Berthou, D. C. Schmidt, and T. Gayraud, “Software-defined networking: Challenges and research opportunities for future internet,” Computer Networks,vol. 75, 2014.
9. OpenFlow Switch Specification [Електронний ресурс] // Open Networking Foundation – Режим доступу до ресурсу: https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.4.0.pdf

10. Лихачев В.А. Программно-конфигурируемые сети на основе протокола OpenFlow [Електронний ресурс] // Вістник ВГУ, Серія: Системний аналіз і інформаційні технології №1, 2014. – Режим доступу до ресурсу: http://www.vestnik.vsu.ru/pdf/analiz/2014/01/2014-01-10.pdf
11. P. Goransson, C. Black, and T. Culver, Software Defined Networks, Second Edition: A Comprehensive Approach, 2nd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016.
12. S. J. Samuel, K. Rvp, K. Sashidhar, and C. R. Bharathi, “a Survey on Big Data and Its Research Challenges,” ARPN Journal of Engineering and Applied Sciences, vol. 10, no. 8, 2015.
13. D. Laney, “3D Data Management: Controlling Data Volume, Velocity, and Variety,” Application Delivery Strategies by META Group Inc., vol. 949, p. 4, 2001.
14. Big Data [Електронний ресурс] // IT Enterprise, 2018. - Режим доступу до ресурсу: https://www.it.ua/ru/knowledge-base/technology-innovation/big-data-bolshie-dannye
15. A. D. Meshram, A. S. Kulkarni, and S. S. Hippargi, “Big Data Analytics using Real-Time Architecture,” International Journal of Latest Trendes in Engineering and Technology(IJLTET), vol. 6, no. 4, 2016.
16. J. A. Stankovic, S. H. Son, and J. Hansson, “Misconceptions About Real-Time Databases,” Computer, vol. 32, 1999.
17. K. Grolinger, M. Hayes, W. A. Higashino, A. L’Heureux, D. S. Allison, and M. A. Capretz, “Challenges for MapReduce in Big Data,” in 2014 IEEE World Congress on Services, pp. 182–189, IEEE, Jun. 2014.
18. N. Marz and J. Warren, Big Data - Principles and Best Practices of Scalable Real-Time Data Systems. Manning Publications Co., 2015.
19. D.Awduche, A. Chiu, A.Elwalid, I.Widjaja “Overview and Principles of Internet Traffic Engineering”, RFC 3784, 2002.

20. Z. Shu, J. Wan, J. Lin, S. Wang, D. Li, S. Rho, and C. Yang, “Traffic engineering in software-defined networking: Measurement and management,” IEEE Access, vol. 4, 2016.
21. B. Niven-Jenkins, D. Brungard, M. Betts, N. Sprecher, and S. Ueno, “Requirements of an MPLS Transport Profile,” RFC 5654, 2009.
22. E. B. Claise, “Cisco systems netflow services export version 9,” RFC 3954, RFC Editor, 2014.
23. sFlow [Електронний ресурс] // Режим доступу до ресурсу: http://sflow.org/about/index.php
24. N. L. M. Van Adrichem, C. Doerr, and F. A. Kuipers, “OpenNetMon: Network monitoring in OpenFlow software-defined networks,” IEEE/IFIP NOMS 2014 - IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World, 2014.
25. X. T. Phan and K. Fukuda, “SDN-Mon: Fine-Grained Traffic Monitoring Framework in Software-Defined Networks,” Journal of Information Processing, vol. 25, no. 0, 2017.
26. P. H. A. Rezende, P. R. S. L. Coelho, L. F. Faina, L. J. Camargos, and R. Pasquini, “Analysis of monitoring and multipath support on top of OpenFlow specification,” International Journal of Network Management, vol. 28, p. e2017, May 2018.
27. M. Malboubi, S.-M. Peng, P. Sharma, and C.-N. Chuah, “A learning-based measurement framework for traffic matrix inference in software defined networks,” Computers & Electrical Engineering, vol. 66, Feb. 2018.
28. M. Polverini, A. Baiocchi, A. Cianfrani, A. Iacovazzi, and M. Listanti, “The Power of SDN to Improve the Estimation of the ISP Traffic Matrix Through the Flow Spread Concept,” IEEE Journal on Selected Areas in Communications, vol. 34, pp. 1904–1913, Jun. 2016.

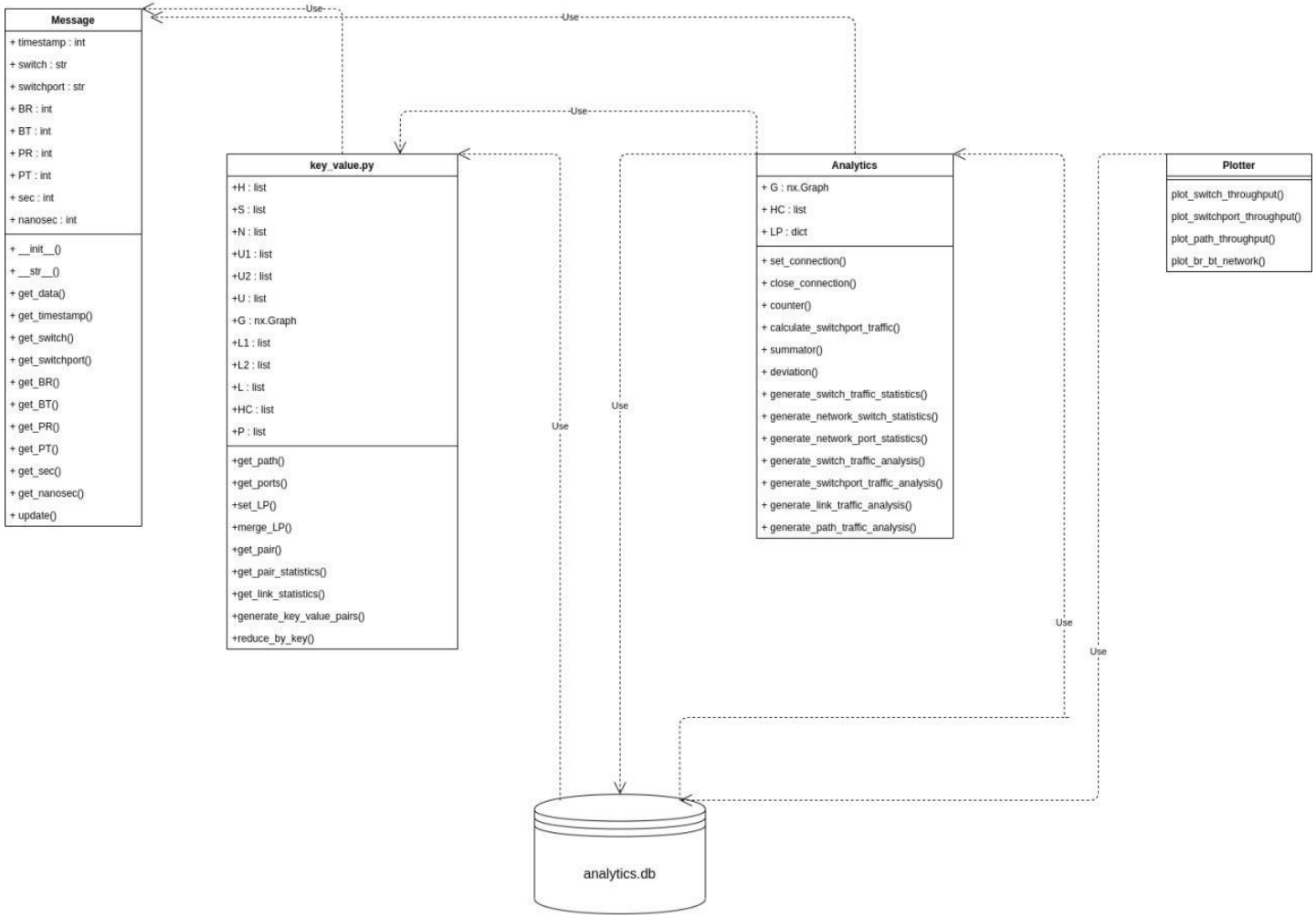
29. W. Queiroz, M. A. Capretz, and M. Dantas, “An approach for SDN traffic monitoring based on big data techniques,” Journal of Network and Computer Applications, vol. 131, Apr. 2019.

ДОДАТОК 1. Структурна схема пристрою



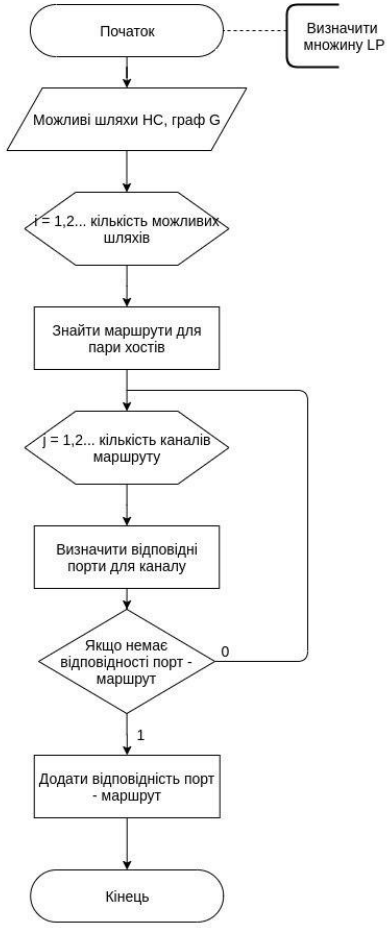
					ІАЛЦ466454.004.Д1			
Зм.	Арк.	№ документа	Підпис	Дата	ДОДАТОК 1. Структурна схема пристрою	Літера	Маса	Масштаб
Розробив		Бабко Д. С.						
Перевірив		Калюжний О. О.				Лист 1		Листів 1
Т. контр								
Н.контр.		Сімоненко В. П.				ФІОТ Гр. ІО-63		
Затв.								

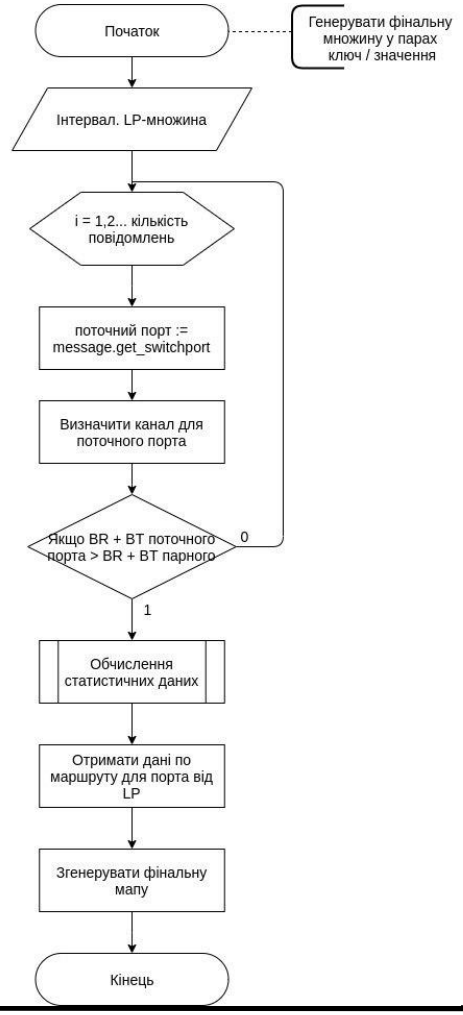
ДОДАТОК 2. Функціональна схема



					ІАЛЦ466454.005.Д2		
Зм.	Арк.	№ документа	Підпис	Дата	ДОДАТОК 2. Функціональна схема	Літера	Маса / Масштаб
Розробив		Бабко Д. С.					
Перевірив		Калюжний О. О.				Лист 1	Листів 1
Т. контр							
Н.контр.		Сімоненко В. П.				ФІОТ Гр. ІО-63	
Затв.							

ДОДАТОК 3. Принципова схема апаратного забезпечення





					ІАЛЦ466454.006.ДЗ		
Зм.	Арк.	№ документа	Підпис	Дата	ДОДАТОК 3. Принципова схема апаратного забезпечення	Літера	Маса / Масштаб
Розробив		Бабко Д. С.					
Перевірив		Калюжний О. О.				Лист 1	Листів 1
Т. контр							
Н.контр.		Сімоненко В. П.				ФІОТ Гр. ІО-63	
Затв.							

ДОДАТОК 4. ПРОГРАМНИЙ КОД ДОДАТКУ

```
from message import Message
import sqlite3
import networkx as nx
import matplotlib.pyplot as plt

def get_path(G, h_start, h_end):
  return [i for i in nx.all_shortest_paths(G, h_start, h_end)]
def get_ports(el1, el2):
  linkports = []
  for elem in L1:
    if el1 == elem[0] and el2 == elem[1]:
      linkports.append(elem[2])
      linkports.append(None)
      return linkports
  for elem in L2:
    if el1 == elem[0] and el2 == elem[2]:
      linkports.append(elem[1])
      linkports.append(elem[3])
      return linkports
  return False
def set_LP():
  LP = set()
  for start_end in HC:
    pathes = get  path(G, start_end[0], start_end[-1])
    for path in pathes:
      for index in range(len(path) - 1):
        ports = get_ports(path[index], path[index + 1])
        if ports:
          if ports[0] not in LP:
            LP.add((ports[0], start_end))
          else:
            LP.update((ports[0], start_end)
```

```
            if ports[1] is not None:
                if ports[1] not in LP:
                    LP.add((ports[1], start_end))
                else:
                    LP.update((ports, start_end))
    return LP
def merge_lp(LP):
    result = dict()
    for elem_of_set in LP:
        ways = []
        if not result.get(elem_of_set[0], False):
            for elem_of_set_second_round in LP:
                if elem_of_set_second_round[0] == elem_of_set[0] and
elem_of_set_second_round[1] not in ways:
                    ways.append(elem_of_set_second_round[1])
        else:
            continue
        result[elem_of_set[0]] = ways
    return result
def generate_messages(messages):
        for message in messages:
            message.udpate()
    else:
        for switchport in LP.keys():
            messages.append(Message(switchport))
def set_connection():
    connection = sqlite3.connect("analytics.db")
    cursor = connection.cursor()
    return cursor
def close_connection(cursor):
    cursor.connection.commit()
    cursor.connection.close()
def counter(func):
    """
    Декоратор, считающий и выводящий количество вызовов
    декорируемой функции.
    """

    func.__invocation_count__ = 0
```

```
  def wrapper(*args, **kwargs):
      func.__invocation_count__ += 1
      res = func(*args, **kwargs)
      print("{0} была вызвана: {1}x".format(func.__name__,
func.__invocation_count__))
      return res

  return wrapper
@counter
def calculate_switchport_traffic(messages, id):
  cursor = set_connection()
  for message in messages:
      switchport = message.switchport
      switch = message.switch
      pr = (message.PR - message.get_previous_pr())
      pt = (message.PT - message.get_previous_pt())
      br = (message.BR - message.get_previous_br())
      bt = (message.BT - message.get_previous_bt())
      # pr = message.PR
      # pt = message.PT
      # br = message.BR
      # bt = message.BT
      sec = (message.sec - message.get_previous_sec())
      pr_throughput = pr / sec
      pt_throughput = pt / sec
      packet_troughput = pr_throughput + pt_throughput
      br_throughput = br / sec
      bt_throughput = bt / sec
      bytes_throughput = br_throughput + bt_throughput
      timestamp = int(f"{message.timestamp:10}")
      values = [id, timestamp, switchport, switch, pr, pt, br, bt,
            pr_throughput, pt_throughput, packet_troughput, bt_throughput,
br  throughput, bytes  throughput]
      query = "INSERT INTO switchport_statistics values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?)"
      try:
         cursor.execute(query, values)
         id += 1
      except sqlite3.IntegrityError as e:
         pass
```

```
  close_connection(cursor)
def summator(*args):
  return sum([x for x in args])
def deviation(*args):
  arg = [arg for arg in args[:-1]]
  m = args[-1]
  dev = 0
  for i in arg:
    dev += (i - m) ** 2
  dev = sqrt(dev / len(arg))
  return dev
def generate_switch_traffic_statistics():
  id = 1
  cursor = set  connection()
  cursor.execute("delete from switch_statistics")
  cursor.execute("select switch from switchport_statistics")
  for switch in cursor.fetchall():
    timestamp_visited = set()
    if switch in cursor.execute("select switch from switch_statistics where switch
= ?", switch).fetchall():
      continue
    else:
      cursor.execute("select timestamp from switchport_statistics where switch =
?", switch)
      for timestamp in cursor.fetchall():
        if timestamp in timestamp_visited:
          continue
        else:
          timestamp_visited.add(timestamp)
          cursor.execute("select PR, PT, BR, BT, PRT, PTT, PTr, BRT, BTT,
BTr from switchport_statistics "
                         "where switch = ? and timestamp = ?", (switch[0],
timestamp[0]))
          values_for_all_ports_in_switch = cursor.fetchall()
          summary = list(map(summator, *values_for_all_ports_in_switch))
          mean_for_switch = list(map(lambda x: 1 /
len(values_for_all_ports_in_switch) * x, summary))
          deviation_for_switch = list(map(deviation,
*values_for_all_ports_in_switch, mean_for_switch))
          query = (
```

```
                "INSERT INTO switch_statistics values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,"
                "?, ?, ?, ?, ?, ?, ?, ?, ?, ?)")
            values = [id, timestamp[0], switch[0]] + summary + mean_for_switch +
deviation_for_switch
            try:
                cursor.execute(query, values)
                id += 1
            except sqlite3.IntegrityError as e:
                pass
    close_connection(cursor)
def generate_network_switch_statistics():
    cursor = set_connection()
    cursor.execute("delete from network_switch_statistics")
    visited_timestamps = set()
    cursor.execute("select timestamp from switch_statistics")
    id = 1
    for timestamp in cursor.fetchall():
        # print(timestamp)
        if timestamp[0] in visited_timestamps:
            continue
        else:
            visited  timestamps.add(timestamp[0])
            cursor.execute("select TPR, TPT, TBR, TBT, TPRT, TPTT, TPTr, TBRT,
TBTT, TBTr "
                           "from switch_statistics where timestamp = ?", timestamp)
            values_to_map = cursor.fetchall()
            summary = list(map(summator, *values  to  map))
            mean_for_network = list(map(lambda x: 1 / len(values_to_map) * x,
summary))
            deviation_for_network = list(map(deviation, *values_to_map,
mean_for_network))
            query = (
                "INSERT INTO network_switch_statistics values (?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,"
                "?, ?, ?, ?, ?, ?, ?, ?, ?)")
            values = [id, timestamp[0]] + summary + mean_for_network +
deviation  for_network
            try:
                cursor.execute(query, values)
```

```
            id += 1
        except sqlite3.IntegrityError as e:
            pass
    close_connection(cursor)
def generate_network_port_statistics():
    cursor = set_connection()
    cursor.execute("delete from network_port_statistics")
    visited_timestamps = set()
    id = 1
    cursor.execute("select timestamp from switchport_statistics")
    for timestamp in cursor.fetchall():
        if timestamp[0] in visited_timestamps:
            continue
        else:
            visited_timestamps.add(timestamp[0])
            cursor.execute("select PR, PT, BR, BT, PRT, PTT, PTr, BRT, BTT, BTr "
                           "from switchport_statistics where timestamp = ?", timestamp)
            values_to_map = cursor.fetchall()
            summary = list(map(summator, *values_to_map))
            mean_for_network = list(map(lambda x: 1 / len(values_to_map) * x,
summary))
            deviation_for_network = list(map(deviation, *values_to_map,
mean_for_network))
            query = (
                "INSERT INTO network_port_statistics values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?, ?, ?, ?, ?, "
                "?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)")
            values = [id, timestamp[0]] + summary + mean_for_network +
deviation_for_network
            try:
                cursor.execute(query, values)
                id += 1
            except sqlite3.IntegrityError as e:
                pass
    close_connection(cursor)
    return True
def generate_switch_traffic_analysis():
    result = dict()
    cursor = set_connection()
    cursor.execute(
```

```
        "select * from network_switch_statistics")
    for row in cursor.fetchall():
        timestamp = row[1],
        means = row[12:22]
        deviations = row[22:]
        cursor.execute("select switch, TPR, TPT, TBR, TBT, TPRT, TPTT, TPTr,
TBRT, TBTT, TBTR "
                       "from switch_statistics where timestamp = ?", timestamp)
        for another  row in cursor.fetchall():
            z_values = list(map(lambda x, m, d: (x - m) / d, another_row[1:], means,
deviations))
            result[timestamp[0], another_row[0]] = z_values
    close_connection(cursor)
    return result
def generate_switchport_traffic_analysis():
    result = dict()
    cursor = set_connection()
    cursor.execute(
        "select timestamp, MeanPR, MeanPT, MeanBR, MeanBT, MeanPRT,
MeanPTT, MeanPTr, MeanBRT, MeanBTT, MeanBTR,"
        "DeviationPR, DeviationPT, DeviationBR, DeviationBT, DeviationPRT,
DeviationPTT, DeviationPTr,"
        "DeviationBRT, DeviationBTT, DeviationBTr from
network_port_statistics")
    for row in cursor.fetchall():
        timestamp = row[0],
        means = row[1:11]
        deviations = row[11:]
        cursor.execute("select switchport, PR, PT, BR, BT, PRT, PTT, PTr, BRT,
BTT, BTR "
                       "from switchport_statistics where timestamp = ?", timestamp)
        for another_row in cursor.fetchall():
            z_values = list(map(lambda x, m, d: (x - m) / d, another_row[1:], means,
deviations))
            result[timestamp[0], another_row[0]] = z_values
    close_connection(cursor)
    return result
# LINK AND PATH STATISTICS
def generate_link_traffic_analysis():
    result = dict()
```

```
    taken_switchports = set()
    cursor = set_connection()
    cursor.execute("select timestamp, switchport, BR, BT from
switchport_statistics")
    for row in cursor.fetchall():
        if row[:2] not in taken_switchports:
            timestamp = row[0]
            switchport = row[1]
            statistics = row[2:]
            pair = get  pair(switchport)
            cursor.execute("select BR, BT from switchport_statistics where switchport
= ? and timestamp = ?", (pair, timestamp))
            statistics_for_pair = cursor.fetchall()
            query = "select PR, PT, BR, BT, PRT, PTT, PTr, BRT, BTT, BTr from
switchport_statistics where switchport = ? and timestamp = ?"
            if sum(statistics) >= sum(*statistics_for_pair):
                values = (switchport, timestamp)
            else:
                values = (pair, timestamp)
            taken_switchports.add((timestamp, values[0]))
            cursor.execute(query, values)
            statistics_for_link = cursor.fetchall()
            result[timestamp, values[0]] = statistics_for_link[0]
        else:
            continue
    close_connection(cursor)
    return result
def generate_path_traffic_analysis(link_traffic_analysis, switchport_traffic_analysis):
    id = 1
    cursor = set_connection()
    cursor.execute("delete from path_traffic_analysis")
    visited_switchports = set()
    cursor.execute("select timestamp, switchport, switch from
switchport_statistics")
    for row in cursor.fetchall():
        switchport = row[1]
        switch = row[-1]
        path_list = LP.get(switchport)
        timestamp = row[0]
        pair = get_pair(switchport)
```

```
        if link_traffic_analysis.get((timestamp, switchport)):
            link = switchport
        else:
            link = pair
        statistics = link_traffic_analysis.get((timestamp, link))
        z_values = switchport_traffic_analysis.get((int(timestamp), link))
        query = ("INSERT INTO path_traffic_analysis values (?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)")
        for el in path  list:
            if pair[:2] == "s1":
                switch = "s1"
            elif pair[:2] == "s2":
                switch = "s2"
            path = f"{switch}/{el[0]}/{el[1]}"
            values = [id, path, timestamp] + [x for x in statistics] + z_values
            if (path, timestamp) in visited_switchports:
                continue
            try:
                cursor.execute(query, values)
                id += 1
            except sqlite3.IntegrityError as e:
                pass
            visited_switchports.add((path, timestamp))
    close_connection(cursor)
```